Chapter 2

Neighborhood-Based Collaborative Filtering

"*When one neighbor helps another, we strengthen our communities.*" – Jennifer Pahlka

2.1 Introduction

Neighborhood-based collaborative filtering algorithms, also referred to as *memory-based algorithms*, were among the earliest algorithms developed for collaborative filtering. These algorithms are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings. There are two primary types of neighborhood-based algorithms:

1. *User-based collaborative filtering:* In this case, the ratings provided by similar users to a *target* user A are used to make recommendations for A. The predicted ratings of A are computed as the weighted average values of these "peer group" ratings for each item.

2. *Item-based collaborative filtering:* In order to make recommendations for *target* item B, the first step is to determine a set S of items, which are most similar to item B. Then, in order to predict the rating of any particular user A for item B, the ratings in set S, which are specified by A, are determined. The weighted average of these ratings is used to compute the predicted rating of user A for item B.

An important distinction between user-based collaborative filtering and item-based collaborative filtering algorithms is that the ratings in the former case are predicted using the ratings of neighboring *users*, whereas the ratings in the latter case are predicted using

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_2

the user's *own* ratings on neighboring (i.e., closely related) *items*. In the former case, neighborhoods are defined by similarities among users (rows of ratings matrix), whereas in the latter case, neighborhoods are defined by similarities among items (columns of ratings matrix). Thus, the two methods share a complementary relationship. Nevertheless, there are considerable differences in the types of recommendations that are achieved using these two methods.

For the purpose of subsequent discussion, we assume that the user-item ratings matrix is an incomplete $m \times n$ matrix $R = [r_{uj}]$ containing m users and n items. It is assumed that only a small subset of the ratings matrix is specified or observed. Like all other collaborative filtering algorithms, neighborhood-based collaborative filtering algorithms can be formulated in one of two ways:

1. *Predicting the rating value of a user-item combination:* This is the simplest and most primitive formulation of a recommender system. In this case, the missing rating r_{uj} of the user u for item j is predicted.

2. *Determining the top-k items or top-k users:* In most practical settings, the merchant is not necessarily looking for specific ratings values of user-item combinations. Rather, it is more interesting to learn the top-k most relevant items for a particular user, or the top-k most relevant users for a particular item. The problem of determining the top-k items is more common than that of finding the top-k users. This is because the former formulation is used to present lists of recommended items to users in Web-centric scenarios. In traditional recommender algorithms, the "top-k problem" almost always refers to the process of finding the top-k items, rather than the top-k users. However, the latter formulation is also useful to the merchant because it can be used to determine the best users to target with marketing efforts.

The two aforementioned problems are closely related. For example, in order to determine the top-k items for a particular user, one can predict the ratings of each item for that user. The top-k items can be selected on the basis of the predicted rating. In order to improve efficiency, neighborhood-based methods pre-compute some of the data needed for prediction in an offline phase. This pre-computed data can be used in order to perform the ranking in a more efficient way.

This chapter will discuss various neighborhood-based methods. We will study the impact of some properties of ratings matrices on collaborative filtering algorithms. In addition, we will study the impact of the ratings matrix on recommendation effectiveness and efficiency. We will discuss the use of clustering and graph-based representations for implementing neighborhood-based methods. We will also discuss the connections between neighborhood methods and regression modeling techniques. Regression methods provide an optimization framework for neighborhood-based methods. In particular, the neighborhood-based method can be shown to be a heuristic approximation of a least-squares regression model [72]. This approximate equivalence will be shown in section 2.6. Such an optimization framework also paves the way for the integration of neighborhood methods with other optimization models, such as latent factor models. The integrated approach is discussed in detail in section 3.7 of Chapter 3.

This chapter is organized as follows. Section 2.2 discusses a number of key properties of ratings matrices. Section 2.3 discusses the key algorithms for neighborhood-based collaborative filtering algorithms. Section 2.4 discusses how neighborhood-based algorithms can be made faster with the use of clustering methods. Section 2.5 discusses the use of dimensionality reduction methods for enhancing neighborhood-based collaborative filtering algorithms.

An optimization modeling view of neighborhood-based methods is discussed in section 2.6. A linear regression approach is used to simulate the neighborhood model within a learning and optimization framework. Section 2.7 discusses how graph-based representations can be used to alleviate the sparsity problem in neighborhood methods. The summary is provided in section 2.8.

2.2 Key Properties of Ratings Matrices

As discussed earlier, we assume that the ratings matrix is denoted by R, and it is an $m \times n$ matrix containing m users and n items. Therefore, the rating of user u for item j is denoted by r_{uj}. Only a small subset of the entries in the ratings matrix are typically specified. The specified entries of the matrix are referred to as the training data, whereas the unspecified entries of the matrix are referred to as the test data. This definition has a direct analog in classification, regression, and semisupervised learning algorithms [22]. In that case, all the unspecified entries belong to a special column, which is known as the class variable or dependent variable. Therefore, the recommendation problem can be viewed as a generalization of the problem of classification and regression.

Ratings can be defined in a variety of ways, depending on the application at hand:

1. *Continuous ratings:* The ratings are specified on a continuous scale, corresponding to the level of like or dislike of the item at hand. An example of such a system is the Jester joke recommendation engine [228, 689], in which the ratings can take on any value between -10 and 10. The drawback of this approach is that it creates a burden on the user of having to think of a real value from an infinite number of possibilities. Therefore, such an approach is relatively rare.

2. *Interval-based ratings:* In interval-based ratings, the ratings are often drawn from a 5-point or 7-point scale, although 10-point and 20-point scales are also possible. Examples of such ratings could be numerical integer values from 1 to 5, from -2 to 2, or from 1 to 7. An important assumption is that the numerical values explicitly define the distances between the ratings, and the rating values are typically equidistant.

3. *Ordinal ratings:* Ordinal ratings are much like interval-based ratings, except that ordered *categorical* values may be used. Examples of such ordered categorical values might be responses such as "Strongly Disagree," "Disagree," "Neutral," "Agree," and "Strongly Agree." A major difference from interval-based ratings is that it is not assumed that the difference between any pair of adjacent ratings values is the same. However, in practice, this difference is only theoretical, because these different ordered categorical values are often assigned to equally spaced utility values. For example, one might assign the "Strongly Disagree" response to a rating value of 1, and the "Strongly Agree" response to a rating value of 5. In such cases, ordinal ratings are almost equivalent to interval-based ratings. Generally, the numbers of positive and negative responses are equally balanced in order to avoid bias. In cases where an even number of responses are used, the "Neutral" option is not present. Such an approach is referred to as the *forced choice* method because the neutral option is not present.

4. *Binary ratings:* In the case of binary ratings, only two options are present, corresponding to positive or negative responses. Binary ratings can be considered a special case of both interval-based and ordinal ratings. For example, the Pandora Internet radio station provides users with the ability to either like or dislike a particular music track.

Binary ratings are an example of the case where forced choice is imposed on the user. In cases where the user is neutral, she will often not specify a rating at all.

5. *Unary ratings:* Such systems allow the user to specify a positive preference for an item, but there is no mechanism to specify a negative preference. This is often the case in many real-world settings, such as the use of a "*like*" button on Facebook. More often, unary ratings are derived from customer *actions*. For example, the act of a customer buying an item can be considered a positive vote for an item. On the other hand, if the customer has not bought the item, then it does not necessarily indicate a dislike for the item. Unary ratings are special because they simplify the development of specialized models in these settings.

It is noteworthy that the indirect derivation of unary ratings from customer actions is also referred to as *implicit feedback*, because the customer does not explicitly provide feedback. Rather, the feedback is inferred in an implicit way through the customer's actions. Such types of "ratings" are often easier to obtain because users are far more likely to interact with items on an online site than to explicitly rate them. The setting of implicit feedback (i.e., unary ratings) is inherently different, as it can be considered the matrix completion analog of the positive-unlabeled (PU) learning problem in classification and regression modeling.

The distribution of ratings among items often satisfies a property in real-world settings, which is referred to as the *long-tail* property. According to this property, only a small fraction of the items are rated frequently. Such items are referred to as *popular* items. The vast majority of items are rated rarely. This results in a highly skewed distribution of the underlying ratings. An example of a skewed rating distribution is illustrated in Figure 2.1. The X-axis shows the index of the item in order of decreasing frequency, and the Y-axis shows the frequency with which the item was rated. It is evident that most of the items are rated only a small number of times. Such a rating distribution has important implications for the recommendation process:

1. In many cases, the high-frequency items tend to be relatively competitive items with little profit for the merchant. On the other hand, the lower frequency items have larger profit margins. In such cases, it may be advantageous to the merchant to recommend lower frequency items. In fact, analysis suggests [49] that many companies, such as Amazon.com, make most of their profit by selling items in the long tail.

2. Because of the rarity of observed ratings in the long tail it is generally more difficult to provide robust rating predictions in the long tail. In fact, many recommendation algorithms have a tendency to suggest popular items rather than infrequent items [173]. This phenomenon also has a negative impact on diversity, and users may often become bored by receiving the same set of recommendations of popular items.

3. The long tailed distribution implies that the items, which are frequently rated by users, are fewer in number. This fact has important implications for neighborhood-based collaborative filtering algorithms because the neighborhoods are often defined on the basis of these frequently rated items. In many cases, the ratings of these high-frequency items are not representative of the low-frequency items because of the inherent differences in the rating patterns of the two classes of items. As a result, the prediction process may yield misleading results. As we will discuss in section 7.6 of Chapter 7, this phenomenon can also cause misleading *evaluations* of recommendation algorithms.

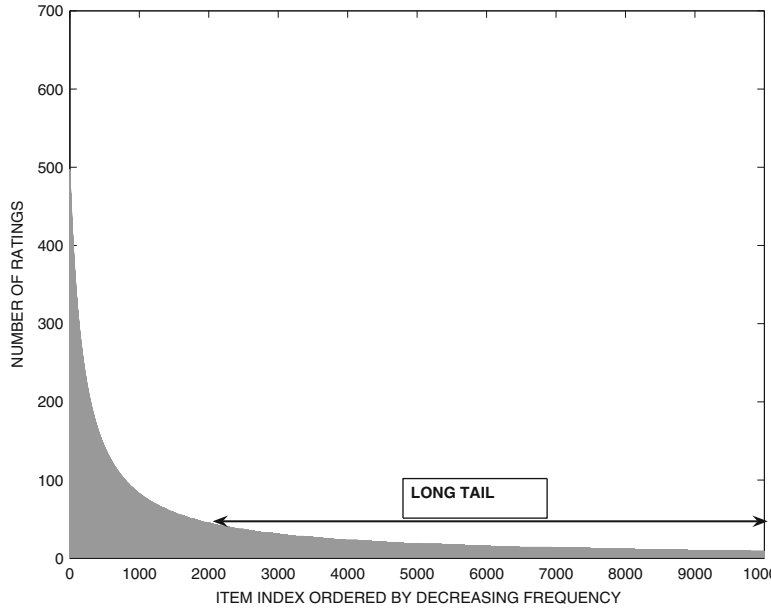


Figure 2.1: The long tail of rating frequencies

Important characteristics of ratings, such as sparsity and the long tail, need to be taken into account during the recommendation process. By adjusting the recommendation algorithms to take such real-world properties into account, it is possible to obtain more meaningful predictions [173, 463, 648].

2.3 Predicting Ratings with Neighborhood-Based Methods

The basic idea in neighborhood-based methods is to use either user-user similarity or item-item similarity to make recommendations from a ratings matrix. The concept of a *neighborhood* implies that we need to determine either similar users or similar items in order to make predictions. In the following, we will discuss how neighborhood-based methods can be used to predict the ratings of specific user-item combinations. There are two basic principles used in neighborhood-based models:

1. *User-based models:* Similar users have similar ratings on the same item. Therefore, if Alice and Bob have rated movies in a similar way in the past, then one can use Alice's observed ratings on the movie *Terminator* to predict Bob's unobserved ratings on this movie.

2. *Item-based models:* Similar items are rated in a similar way by the same user. Therefore, Bob's ratings on similar science fiction movies like *Alien* and *Predator* can be used to predict his rating on *Terminator*.

Since the collaborative filtering problem can be viewed as a generalization of the classification/regression modeling problem, neighborhood-based methods can be viewed as generalizations of nearest neighbor classifiers in the machine learning literature. Unlike

Table 2.1: User-user similarity computation between user 3 and other users

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6	Mean Rating	Cosine$(i, 3)$ (user-user)	Pearson$(i, 3)$ (user-user)
1	7	6	7	4	5	4	5.5	0.956	0.894
2	6	7	?	4	3	4	4.8	0.981	0.939
3	?	3	3	1	1	?	2	1.0	1.0
4	1	2	2	3	3	4	2.5	0.789	-1.0
5	1	?	1	2	3	3	2	0.645	-0.817

Table 2.2: Ratings matrix of Table 2.1 with mean-centering for adjusted cosine similarity computation among items. The adjusted cosine similarities of items 1 and 6 with other items are shown in the last two rows.

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	-1	0	1	1
Cosine$(1, j)$ (item-item)	1	0.735	0.912	-0.848	-0.813	-0.990
Cosine$(6, j)$ (item-item)	-0.990	-0.622	-0.912	0.829	0.730	1

classification, where the nearest neighbors are always determined only on the basis of row similarity, it is possible to find the nearest neighbors in collaborative filtering on the basis of either rows or columns. This is because all missing entries are concentrated in a single column in classification, whereas the missing entries are spread out over the different rows and columns in collaborative filtering (cf. section 1.3.1.3 of Chapter 1). In the following discussion, we will discuss the details of both user-based and item-based neighborhood models, together with their natural variations.

2.3.1 User-Based Neighborhood Models

In this approach, user-based neighborhoods are defined in order to identify similar users to the *target* user for whom the rating predictions are being computed. In order to determine the neighborhood of the target user i, her similarity to all the other users is computed. Therefore, a similarity function needs to be defined between the ratings specified by users. Such a similarity computation is tricky because different users may have different scales of ratings. One user might be biased toward liking most items, whereas another user might be biased toward not liking most of the items. Furthermore, different users may have rated different items. Therefore, mechanisms need to be identified to address these issues.

For the $m \times n$ ratings matrix $R = [r_{uj}]$ with m users and n items, let I_u denote the set of item indices for which ratings have been specified by user (row) u. For example, if the ratings of the first, third, and fifth items (columns) of user (row) u are specified (observed)

and the remaining are missing, then we have $I_u = \{1, 3, 5\}$. Therefore, the set of items rated by both users u and v is given by $I_u \cap I_v$. For example, if user v has rated the first four items, then $I_v = \{1, 2, 3, 4\}$, and $I_u \cap I_v = \{1, 3, 5\} \cap \{1, 2, 3, 4\} = \{1, 3\}$. It is possible (and quite common) for $I_u \cap I_v$ to be an empty set because ratings matrices are generally sparse. The set $I_u \cap I_v$ defines the mutually observed ratings, which are used to compute the similarity between the uth and vth users for neighborhood computation.

One measure that captures the similarity $\text{Sim}(u, v)$ between the rating vectors of two users u and v is the Pearson correlation coefficient. Because $I_u \cap I_v$ represents the set of item indices for which both user u and user v have specified ratings, the coefficient is computed only on this set of items. The first step is to compute the mean rating μ_u for each user u using her specified ratings:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \ldots m\} \tag{2.1}$$

Then, the Pearson correlation coefficient between the rows (users) u and v is defined as follows:

$$\text{Sim}(u, v) = \text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.2}$$

Strictly speaking, the traditional definition of $\text{Pearson}(u, v)$ mandates that the values of μ_u and μ_v should be computed *only* over the items that are rated *both* by users u and v. Unlike Equation 2.1, such an approach will lead to a different value of μ_u, depending on the choice of the other user v to which the Pearson similarity is being computed. However, it is quite common (and computationally simpler) to compute each μ_u just once for each user u, according to Equation 2.1. It is hard to make an argument that one of these two ways of computing μ_u always provides strictly better recommendations than the other. In extreme cases, where the two users have only one mutually specified rating, it can be argued that using Equation 2.1 for computing μ_u will provide more informative results, because the Pearson coefficient will be indeterminate over a single common item in the traditional definition. Therefore, we will work with the simpler assumption of using Equation 2.1 in this chapter. Nevertheless, it is important for the reader to keep in mind that many implementations of user-based methods compute μ_u and μ_v in pairwise fashion during the Pearson computation.

The Pearson coefficient is computed between the target user and all the other users. One way of defining the peer group of the target user would be to use the set of k users with the highest Pearson coefficient with the target. However, since the number of observed ratings in the top-k peer group of a target user may vary significantly with the item at hand, the closest k users are found for the target user separately for each predicted item, such that each of these k users have specified ratings for that item. The weighted average of these ratings can be returned as the predicted rating for that item. Here, each rating is weighted with the Pearson correlation coefficient of its owner to the target user.

The main problem with this approach is that different users may provide ratings on different scales. One user might rate all items highly, whereas another user might rate all items negatively. The raw ratings, therefore, need to be mean-centered in row-wise fashion, before determining the (weighted) average rating of the peer group. The mean-centered rating s_{uj} of a user u for item j is defined by subtracting her mean rating from the raw rating r_{uj}.

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \ldots m\} \tag{2.3}$$

As before, the weighted average of the mean-centered rating of an item in the top-k peer group of target user u is used to provide a *mean-centered* prediction. The mean rating of the target user is then added back to this prediction to provide a *raw* rating prediction $\hat{r}_{uj}$ of target user u for item j. The hat notation "^" on top of r_{uj} indicates a *predicted* rating, as opposed to one that was already observed in the original ratings matrix. Let $P_u(j)$ be the set[1] of k closest users to target user u, who have specified ratings for item j. Users with very low or negative correlations with target user u are sometimes filtered from $P_u(j)$ as a heuristic enhancement. Then, the overall neighborhood-based *prediction function* is as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.4}$$

This broader approach allows for a number of different variations in terms of how the similarity or prediction function is computed or in terms of which items are filtered out during the prediction process.

Example of User-Based Algorithm

Consider the example of Table 2.1. In this case, the ratings of five users $1 \ldots 5$ are indicated for six items denoted by $1 \ldots 6$. Each rating is drawn from the range $\{1 \ldots 7\}$. Consider the case where the target user index is 3, and we want to make item predictions on the basis of the ratings in Table 2.1. We need to compute the predictions $\hat{r}_{31}$ and $\hat{r}_{36}$ of user 3 for items 1 and 6 in order to determine the top recommended item.

The first step is to compute the similarity between user 3 and all the other users. We have shown two possible ways of computing similarity in the last two columns of the same table. The second-last column shows the similarity based on the raw cosine between the ratings and the last column shows the similarity based on the Pearson correlation coefficient. For example, the values of Cosine$(1, 3)$ and Pearson$(1, 3)$ are computed as follows:

$$\text{Cosine}(1, 3) = \frac{6 * 3 + 7 * 3 + 4 * 1 + 5 * 1}{\sqrt{6^2 + 7^2 + 4^2 + 5^2} \cdot \sqrt{3^2 + 3^2 + 1^2 + 1^2}} = 0.956$$

$$\text{Pearson}(1, 3) =$$

$$= \frac{(6 - 5.5) * (3 - 2) + (7 - 5.5) * (3 - 2) + (4 - 5.5) * (1 - 2) + (5 - 5.5) * (1 - 2)}{\sqrt{1.5^2 + 1.5^2 + (-1.5)^2 + (-0.5)^2} \cdot \sqrt{1^2 + 1^2 + (-1)^2 + (-1)^2}}$$

$$= 0.894$$

The Pearson and raw cosine similarities of user 3 with all other users are illustrated in the final two columns of Table 2.1. Note that the Pearson correlation coefficient is much more discriminative and the sign of the coefficient provides information about similarity and dissimilarity. The top-2 closest users to user 3 are users 1 and 2 according to both measures. By using the Pearson-weighted average of the *raw* ratings of users 1 and 2, the following predictions are obtained for user 3 with respect to her unrated items 1 and 6:

$$\hat{r}_{31} = \frac{7 * 0.894 + 6 * 0.939}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{4 * 0.894 + 4 * 0.939}{0.894 + 0.939} = 4$$

[1] In many cases, k valid peers of target user u with observed ratings for item j might not exist. This scenario is particularly common in sparse ratings matrices, such as the case where user u has less than k observed ratings. In such cases, the set $P_u(j)$ will have cardinality less than k.

Thus, item 1 should be prioritized over item 6 as a recommendation to user 3. Furthermore, the prediction suggests that user 3 is likely to be interested in *both* movies 1 and 6 to a greater degree than *any* of the movies she has already rated. This is, however, a result of the bias caused by the fact that the peer group $\{1, 2\}$ of user indices is a far more optimistic group with positive ratings, as compared to the target user 3. Let us now examine the impact of mean-centered ratings on the prediction. The mean-centered ratings are illustrated in Table 2.2. The corresponding predictions with mean-centered Equation 2.4 are as follows:

$$\hat{r}_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} \approx 3.35$$
$$\hat{r}_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} \approx 0.86$$

Thus, the mean-centered computation also provides the prediction that item 1 should be prioritized over item 6 as a recommendation to user 3. There is, however, one crucial difference from the previous recommendation. In this case, the predicted rating of item 6 is only 0.86, which is *less* than all the other items that user 3 has rated. This is a drastically different result than in the previous case, where the predicted rating for item 6 was greater than all the other items that user 3 had rated. Upon visually inspecting Table 2.1 (or Table 2.2), it is indeed evident that item 6 ought to be rated very low by user 3 (compared to her other items), because her closest peers (users 1 and 2) have also rated it lower than their other items. Thus, the mean-centering process enables a much better *relative* prediction with respect to the ratings that have already been observed. In many cases, it can also affect the relative order of the predicted items. The only weakness in this result is that the predicted rating of item 6 is 0.85, which is outside the range of allowed ratings. Such ratings can always be used for ranking, and the predicted value can be corrected to the closest value in the allowed range.

2.3.1.1 Similarity Function Variants

Several other variants of the similarity function are used in practice. One variant is to use the cosine function on the *raw* ratings rather than the mean-centered ratings:

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}} \tag{2.5}$$

In some implementations of the raw cosine, the normalization factors in the denominator are based on all the specified items and not the mutually rated items.

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_v} r_{vk}^2}} \tag{2.6}$$

In general, the Pearson correlation coefficient is preferable to the raw cosine because of the *bias adjustment* effect of mean-centering. This adjustment accounts for the fact that different users exhibit different levels of generosity in their global rating patterns.

The reliability of the similarity function $\text{Sim}(u, v)$ is often affected by the number of common ratings $|I_u \cap I_v|$ between users u and v. When the two users have only a small number of ratings in common, the similarity function should be reduced with a discount factor to de-emphasize the importance of that user pair. This method is referred to as *significance weighting*. The discount factor kicks in when the number of common ratings

between the two users is less than a particular threshold β. The value of the discount factor is given by $\frac{\min\{|I_u \cap I_v|, \beta\}}{\beta}$, and it always lies in the range $[0, 1]$. Therefore, the discounted similarity $\text{DiscountedSim}(u, v)$ is given by the following:

$$\text{DiscountedSim}(u, v) = \text{Sim}(u, v) \cdot \frac{\min\{|I_u \cap I_v|, \beta\}}{\beta} \tag{2.7}$$

The discounted similarity is used both for the process of determining the peer group and for computing the prediction according to Equation 2.4.

2.3.1.2 Variants of the Prediction Function

There are many variants of the prediction function used in Equation 2.4. For example, instead of mean-centering the raw rating r_{uj} to the centered value s_{uj}, one might use the Z-score z_{uj}, which further divides s_{uj} with the standard deviation σ_u of the observed ratings of user u. The standard deviation is defined as follows:

$$\sigma_u = \sqrt{\frac{\sum_{j \in I_u} (r_{uj} - \mu_u)^2}{|I_u| - 1}} \quad \forall u \in \{1 \ldots m\} \tag{2.8}$$

Then, the standardized rating is computed as follows:

$$z_{uj} = \frac{r_{uj} - \mu_u}{\sigma_u} = \frac{s_{uj}}{\sigma_u} \tag{2.9}$$

Let $P_u(j)$ denote the set of the top-k similar users of target user u, for which the ratings of item j have been observed. In this case, the predicted rating $\hat{r}_{uj}$ of target user u for item j is as follows:

$$\hat{r}_{uj} = \mu_u + \sigma_u \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot z_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.10}$$

Note that the weighted average needs to be *multiplied* with σ_u in this case. In general, if a function $g(\cdot)$ is applied during ratings normalization, then its inverse needs to be applied during the final prediction process. Although it is generally accepted that normalization improves the prediction, there seem to be conflicting conclusions in various studies on whether mean-centering or the Z-score provides higher-quality results [245, 258]. One problem with the Z-score is that the predicted ratings might frequently be outside the range of the permissible ratings. Nevertheless, even when the predicted values are outside the range of permissible ratings, they can be used to *rank* the items in order of desirability for a particular user.

A second issue in the prediction is that of the weighting of the various ratings in Equation 2.4. Each mean-centered rating s_{vj} of user v for item j is weighted with the similarity $\text{Sim}(u, v)$ of user v to the target user u. While the value of $\text{Sim}(u, v)$ was chosen to be the Pearson correlation coefficient, a commonly used practice is to *amplify* it by exponentiating it to the power of α. In other words, we have:

$$\text{Sim}(u, v) = \text{Pearson}(u, v)^{\alpha} \tag{2.11}$$

By choosing $\alpha > 1$, it is possible to amplify the importance of the similarity in the weighting of Equation 2.4.

As discussed earlier, neighborhood-based collaborative filtering methods are generalizations of nearest neighbor classification/regression methods. The aforementioned discussion is closer to nearest neighbor regression modeling, rather than nearest neighbor classification, because the predicted value is treated as a continuous variable throughout the prediction process. It is also possible to create a prediction function which is closer to a classification method by treating ratings as categorical values and ignoring the ordering among the ratings. Once the peer group of the target user u has been identified, the number of *votes* for each possible rating value (e.g., Agree, Neutral, Disagree) within the peer group is determined. The rating with the largest number of votes is predicted as the relevant one. This approach has the advantage of providing the most *likely* rating rather than the average rating. Such an approach is generally more effective in cases where the number of distinct ratings is small. It is also useful in the case of ordinal ratings, where the exact distances between pairs of rating values are not defined. In cases where the granularity of ratings is high, such an approach is less robust and loses a lot of ordering information among the ratings.

2.3.1.3 Variations in Filtering Peer Groups

The peer group for a target user may be defined and filtered in a wide variety of ways. The simplest approach is to use the top-k most similar users to the target user as her peer group. However, such an approach might include users that are weakly or negatively correlated with the target. Weakly correlated users might add to the error in the prediction. Furthermore, negatively correlated ratings often do not have as much predictive value in terms of potential inversion of the ratings. Although the prediction function technically allows the use of weak or negative ratings, their use is not consistent with the broader principle of neighborhood methods. Therefore, ratings with weak or negative correlations are often filtered out.

2.3.1.4 Impact of the Long Tail

As discussed in section 2.2, the distribution of ratings typically shows a long-tail distribution in many real scenarios. Some movies may be very popular and they may repeatedly occur as commonly rated items by different users. Such ratings can sometimes worsen the quality of the recommendations because they tend to be less discriminative across different users. The negative impact of these recommendations can be experienced both during the peer group computation and also during the prediction computation (cf. Equation 2.4). This notion is similar in principle to the deterioration in retrieval quality caused by popular and noninformative words (e.g., "a," "an," "the") in document retrieval applications. Therefore, the proposed solutions used in collaborative filtering are also similar to those used in the information retrieval literature. Just as the notion of *Inverse Document Frequency* (idf) exists in the information retrieval literature [400], one can use the notion of *Inverse User Frequency* in this case. If m_j is the number of ratings of item j, and m is the total number of users, then the weight w_j of the item j is set to the following:

$$w_j = \log\left(\frac{m}{m_j}\right) \quad \forall j \in \{1 \ldots n\} \tag{2.12}$$

Each item j is weighted by w_j both during the similarity computation and during the recommendation process. For example, the Pearson correlation coefficient can be modified to include the weights as follows:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{vk} - \mu_v)^2}} \tag{2.13}$$

Item weighting can also be incorporated in other collaborative filtering methods. For example, the final prediction step of item-based collaborative filtering algorithms can be modified to use weights, even though the adjusted cosine similarity between two items remains unchanged by the weights.

2.3.2 Item-Based Neighborhood Models

In item-based models, peer groups are constructed in terms of *items* rather than *users*. Therefore, similarities need to be computed between items (or columns in the ratings matrix). Before computing the similarities between the columns, each row of the ratings matrix is centered to a mean of zero. As in the case of user-based ratings, the average rating of each item in the ratings matrix is subtracted from each rating to create a mean-centered matrix. This process is identical to that discussed earlier (see Equation 2.3), which results in the computation of mean-centered ratings s_{uj}. Let U_i be the indices of the set of users who have specified ratings for item i. Therefore, if the first, third, and fourth users have specified ratings for item i, then we have $U_i = \{1, 3, 4\}$.

Then, the *adjusted* cosine similarity between the items (columns) i and j is defined as follows:

$$\text{AdjustedCosine}(i, j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}} \tag{2.14}$$

This similarity is referred to as the adjusted cosine similarity because the ratings are mean-centered before computing the similarity value. Although the Pearson correlation can also be used on the columns in the case of the item-based method, the adjusted cosine generally provides superior results.

Consider the case in which the rating of target item t for user u needs to be determined. The first step is to determine the top-k most similar *items* to *item* t based on the aforementioned adjusted cosine similarity. Let the top-k matching items to item t, for which the user u has specified ratings, be denoted by $Q_t(u)$. The *weighted* average value of these (raw) ratings is reported as the predicted value. The weight of item j in this average is equal to the adjusted cosine similarity between item j and the target item t. Therefore, the predicted rating $\hat{r}_{ut}$ of user u for target item t is as follows:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j, t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j, t)|} \tag{2.15}$$

The basic idea is to leverage the user's *own* ratings on similar items in the final step of making the prediction. For example, in a movie recommendation system, the item peer group will typically be movies of a similar genre. The ratings history of the *same* user on such movies is a very reliable predictor of the interests of that user.

The previous section discussed a number of variants of the basic approach for user-based collaborative filtering. Because item-based algorithms are very similar to user-based algorithms, similar variants of the similarity function and the prediction function can be designed for item-based methods.

Example of Item-Based Algorithm

In order to illustrate the item-based algorithm, we will use the same example of Table 2.1, which was leveraged to demonstrate the user-based algorithm. The missing ratings of user

3 are predicted with the item-based algorithm. Because the ratings of items 1 and 6 are missing for user 3, the similarity of the columns for items 1 and 6 needs to be computed with respect to the other columns (items).

First, the similarity between items are computed after adjusting for mean-centering. The mean-centered ratings matrix is illustrated in Table 2.2. The corresponding adjusted cosine similarities of each item to 1 and 6, respectively, are indicated in the final two rows of the table. For example, the value of the adjusted cosine between items 1 and 3, denoted by AdjustedCosine$(1, 3)$, is as follows:

$$\text{AdjustedCosine}(1,3) = \frac{1.5 * 1.5 + (-1.5) * (-0.5) + (-1) * (-1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \cdot \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}} = 0.912$$

Other item-item similarities are computed in an exactly analogous way, and are illustrated in the final two rows of Table 2.2. It is evident that items 2 and 3 are most similar to item 1, whereas items 4 and 5 are most similar to item 6. Therefore, the weighted average of the *raw* ratings of user 3 for items 2 and 3 is used to predict the rating $\hat{r}_{31}$ of item 1, whereas the weighted average of the raw ratings of user 3 for items 4 and 5 is used to predict the rating $\hat{r}_{36}$ of item 6:

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$
$$\hat{r}_{36} = \frac{1 * 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$

Thus, the item-based method also suggests that item 1 is more likely to be preferred by user 3 than item 6. However, in this case, because the ratings are predicted using the ratings of user 3 herself, the predicted ratings tend to be much more consistent with the other ratings of this user. As a specific example, it is noteworthy that the predicted rating of item 6 is no longer outside the range of allowed ratings, as in the case of the user-based method. The greater prediction accuracy of the item-based method is its main advantage. In some cases, the item-based method might provide a different set of top-k recommendations, even though the recommended lists will generally be roughly similar.

2.3.3 Efficient Implementation and Computational Complexity

Neighborhood-based methods are always used to determine the best item recommendations for a target user or the best user recommendations for a target item. The aforementioned discussion only shows how to predict the ratings for a particular user-item *combination*, but it does not discuss the actual ranking process. A straightforward approach is to compute all possible rating predictions for the relevant user-item pairs (e.g., all items for a particular user) and then rank them. While this is the basic approach used in current recommender systems, it is important to observe that the prediction process for many user-item combinations reuses many intermediate quantities. Therefore, it is advisable to have an offline phase to store these intermediate computations and then leverage them in the ranking process.

Neighborhood-based methods are always partitioned into an *offline* phase and an *online* phase. In the offline phase, the user-user (or item-item) similarity values and peer groups of the users (or items) are computed. For each user (or item), the relevant peer group is prestored on the basis of this computation. In the online phase, these similarity values and peer groups are leveraged to make predictions with the use of relationships such as Equation 2.4. Let $n' \ll n$ be the maximum number of specified ratings of a user (row), and

$m' \ll m$ be the maximum number of specified ratings of an item (column). Note that n' is the maximum running time for computing the similarity between a pair of users (rows), and m' is the maximum running time for computing the similarity between a pair of items (columns). In the case of user-based methods, the process of determining the peer group of a target user may require $O(m \cdot n')$ time. Therefore, the offline running time for computing the peer groups of all users is given by $O(m^2 \cdot n')$. For item-based methods, the corresponding offline running time is given by $O(n^2 \cdot m')$.

In order to be able to use the approach for varying values of k, one might end up having to store all pairs of nonzero similarities between pairs of users (or items). Therefore, the space requirements of user-based methods are $O(m^2)$, whereas the space requirements of item-based methods are $O(n^2)$. Because the number of users is typically greater than the number of items, the space requirements of user-based methods are generally greater than those of item-based methods.

The online computation of the predicted value according to Equation 2.4 requires $O(k)$ time for both user-based and item-based methods, where k is the size of the user/item neighborhood used for prediction. Furthermore, if this prediction needs to be executed over all items in order to rank them for a target *user*, then the running time is $O(k \cdot n)$ for both user-based and item-based methods. On the other hand, a merchant may occasionally wish to determine the top-r users to be targeted for a specific item. In this case, the prediction needs to be executed over all users in order to rank them for a target *item*, and the running time is $O(k \cdot m)$ for both user-based and item-based methods. It is noteworthy that the primary computational complexity of neighborhood-based methods resides in the offline phase, which needs to be executed occasionally. As a result, neighborhood-based methods tend to be efficient when they are used for online prediction. After all, one can afford to be generous in allocating significantly more computational time to the offline phase.

2.3.4 Comparing User-Based and Item-Based Methods

Item-based methods often provide more relevant recommendations because of the fact that a user's *own* ratings are used to perform the recommendation. In item-based methods, similar *items* are identified to a target item, and the user's own ratings on those items are used to extrapolate the ratings of the target. For example, similar items to a target historical movie might be a set of other historical movies. In such cases, the user's own recommendations for the similar set might be highly indicative of her preference for the target. This is not the case for user-based methods in which the ratings are extrapolated from other users, who might have overlapping but different interests. As a result, item-based methods often exhibit better accuracy.

Although item-based recommendations are often more likely to be accurate, the relative accuracy between item-based and user-based methods also depends on the data set at hand. As you will learn in Chapter 12, item-based methods are also more robust to *shilling attacks* in recommender systems. On the other hand, it is precisely these differences that can lead to greater diversity in the recommendation process for user-based methods over item-based methods. Diversity refers to the fact that the items in the ranked list tend to be somewhat different. If the items are not diverse, then if the user does not like the first item, she might not also like any of the other items in the list. Greater diversity also encourages serendipity, through which somewhat surprising and interesting items are discovered. Item-based methods might sometimes recommend obvious items, or items which are not *novel* from previous user experiences. The notions of novelty, diversity, and serendipity are discussed in detail in Chapter 7. Without sufficient novelty, diversity, and serendipity, users might become bored with very similar recommendations to what they have already watched.

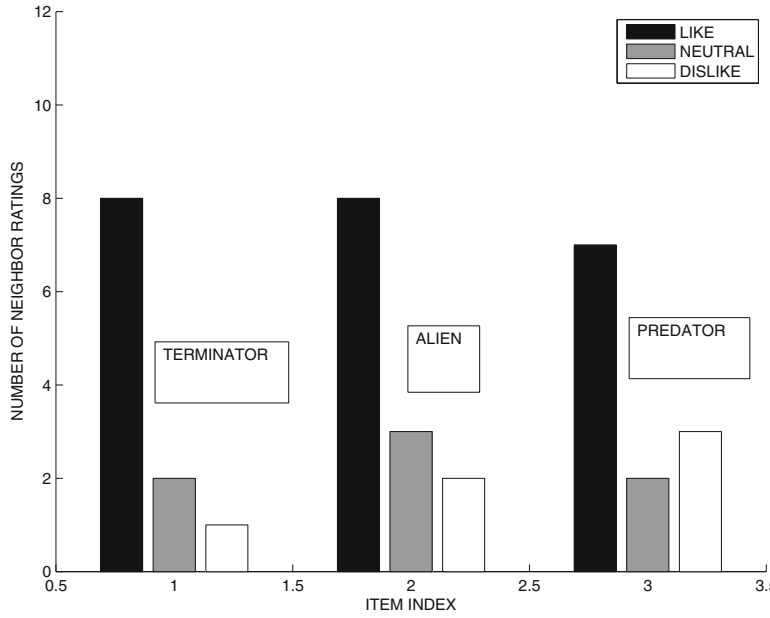


Figure 2.2: Explaining Alice's top recommendations with her neighbor rating histogram

Item-based methods can also provide a concrete reason for the recommendation. For example, Netflix often provides recommendations with statements such as the following:

Because you watched "Secrets of the Wings," [the recommendations are] $\langle List \rangle$.

Such explanations can be concretely addressed with item-based methods[2] by using the item neighborhoods. On the other hand, these explanations are harder to address withuser-based methods, because the peer group is simply a set of anonymous users and not directly usable in the recommendation process.

User-based methods provide different types of explanations. For example, consider a scenario where the movies *Terminator*, *Alien*, and *Predator*, are recommended to Alice. Then, a histogram of her neighbor's ratings for these movies can be shown to her. An example of such a histogram is shown in Figure 2.2. This histogram can be used by Alice to obtain an idea of how much she might like this movie. Nevertheless, the power of this type of explanation is somewhat limited because it does not give Alice an idea of how these movies relate to her *own* tastes or to those of friends she actually knows and trusts. Note that the identity of her neighbors is usually not available to Alice because of privacy concerns.

Finally, item-based methods are more stable with changes to the ratings. This is because of two reasons. First, the number of users is generally much larger than the number of items. In such cases, two users may have a very small number of mutually rated items, but two items are more likely to have a larger number of users who have co-rated them. In the case of user-based methods, the addition of a few ratings can change the similarity values drastically. This is not the case for item-based methods, which are more stable to changes in the values of the ratings. Second, new users are likely to be added more frequently in

[2]The precise method used by Netflix is proprietary and therefore not known. However, item-based methods do provide a viable methodology to achieve similar goals.

commercial systems than new items. In such cases, the computation of neighborhood items can be done only occasionally because item neighborhoods are unlikely to change drastically with the addition of new users. On the other hand, the computation of user neighborhoods needs to be performed more frequently with the addition of new users. In this context, incremental maintenance of the recommendation model is more challenging in the case of user-based methods.

2.3.5 Strengths and Weaknesses of Neighborhood-Based Methods

Neighborhood methods have several advantages related to their simplicity and intuitive approach. Because of the simple and intuitive approach of these methods, they are easy to implement and debug. It is often easy to justify why a specific item is recommended, and the interpretability of item-based methods is particularly notable. Such justifications are often not easily available in many of the model-based methods discussed in later chapters. Furthermore, the recommendations are relatively stable with the addition of new items and users. It is also possible to create incremental approximations of these methods.

The main disadvantage of these methods is that the offline phase can sometimes be impractical in large-scale settings. The offline phase of the user-based method requires at least $O(m^2)$ time and space. This might sometimes be too slow or space-intensive with desktop hardware, when m is of the order of tens of millions. Nevertheless, the online phase of neighborhood methods is always efficient. The other main disadvantage of these methods is their limited coverage because of sparsity. For example, if none of John's nearest neighbors have rated *Terminator*, it is not possible to provide a rating prediction of *Terminator* for John. On the other hand, we care only about the top-k items of John in most recommendation settings. If none of John's nearest neighbors have rated *Terminator*, then it might be evidence that this movie is not a good recommendation for John. Sparsity also creates challenges for robust similarity computation when the number of mutually rated items between two users is small.

2.3.6 A Unified View of User-Based and Item-Based Methods

The respective weaknesses of user-based and item-based methods arise out of the fact that the former ignores the similarity between the columns of the ratings matrix, whereas the latter ignores the similarity between the rows while determining the most similar entries. A natural question arises whether we can determine the most similar *entries* to a target entry by unifying the two methods. By doing so, one does not need to ignore the similarity along either rows or columns. Rather, one can *combine* the similarity information between rows and columns.

In order to achieve this goal, it is crucial to understand that the user-based and item-based methods are almost identical (with some minor differences), once the rows have been mean-centered. We can assume without loss of generality that the rows of the ratings matrix are mean-centered because the mean of each row can be added back to each entry after the prediction. It is also noteworthy that if the rows are mean-centered then the Pearson correlation coefficient between rows is identical[3] to the cosine coefficient. Based on this

[3]There can be some minor differences depending on how the mean is computed for each row within the Pearson coefficient. If the mean for each row is computed using all the observed entries of that row (rather than only the mutually specified entries), then the Pearson correlation coefficient is identical to the cosine coefficient for row-wise mean-centered matrices.

assumption, the user-based and item-based methods can be described in a unified way to predict the entry r_{uj} in the ratings matrix R:

1. For a target entry (u, j) determine the most similar rows/columns of the ratings matrix with the use of the cosine coefficient between rows/columns. For user-based methods rows are used, whereas for item-based methods, columns are used.

2. Predict the target entry (u, j) using a weighted combination of the ratings in the most similar rows/columns determined in the first step.

Note that the aforementioned description ignores *either* the rows or the columns in each step. One can, of course, propose a generalized description of the aforementioned steps in which the similarity and prediction information along rows and columns are *combined*:

1. For a target entry (u, j) determine the most similar *entries* of the ratings matrix with the use of a combination function of the similarity between rows and columns. For example, one can use the sum of the cosine similarity between rows and between columns to determine the most similar entries in the ratings matrix to (u, j).

2. Predict the target entry (u, j) using a weighted combination of the ratings in the most similar *entries* determined in the first step. The weights are based on the similarities computed in the first step.

We have highlighted the steps, which are different in the generalized method. This approach fuses the similarities along rows and columns with the use of a combination function. One can experiment with the use of various combination functions to obtain the most effective results. Detailed descriptions of such unified methods may be found in [613, 622]. This basic principle is also used in the multidimensional model of context-sensitive recommender systems, in which the similarities along users, items, and other contextual dimensions are unified into a single framework (cf. section 8.5.1 of Chapter 8).

2.4 Clustering and Neighborhood-Based Methods

The main problem with neighborhood-based methods is the complexity of the offline phase, which can be quite significant when the number of users or the number of items is very large. For example, when the number of users m is of the order of a few hundred million, the $O(m^2 \cdot n')$ running time of a user-based method will become impractical even for occasional offline computation. Consider the case where $m = 10^8$ and $n' = 100$. In such a case, $O(m^2 \cdot n') = O(10^{18})$ operations will be required. If we make the conservative assumption that each operation requires an elementary machine cycle, a 10GHz computer will require 10^8 seconds, which is approximately 115.74 days. Clearly, such an approach will not be very practical from a scalability point of view.

The main idea of clustering-based methods is to replace the offline nearest-neighbor computation phase with an offline clustering phase. Just as the offline nearest-neighbor phase creates a large number of peer groups, which are centered *at each possible target*, the clustering process creates a smaller number of peer groups which are not necessarily centered at each possible target. The process of clustering is much more efficient than the $O(m^2 \cdot n')$ time required for construction of the peer groups of every possible target. Once the clusters have been constructed, the process of predicting ratings is similar to the approach used in Equation 2.4. The main difference is that the top-k closest peers within the same cluster are used to perform the prediction. It is noteworthy that the pairwise similarity computation

needs to be performed only within the same cluster and therefore, the approach can be significantly more efficient. This efficiency does result in some loss of accuracy because the set of closest neighbors to each target within a cluster is of lower quality than that over the entire data. Furthermore, the clustering granularity regulates the trade-off between accuracy and efficiency. When the clusters are fine-grained, the efficiency improves, but the accuracy is reduced. In many cases, very large gains in efficiency can be obtained for small reductions in accuracy. When the ratings matrices are very large, this approach provides a very practical alternative at a small cost.

One challenge with the use of this approach is the fact that the ratings matrix is incomplete. Therefore, clustering methods need to be adapted to work with massively incomplete data sets. In this context, k-means methods can be easily adapted to incomplete data. The basic idea of a k-means approach is to work with k central points (or "means"), which serve as the representatives of k different clusters. In k-means methods, the solution to a clustering can be fully represented by the specification of these k representatives. Given a set of k representatives $\overline{Y_1} \ldots \overline{Y_k}$, each data point is assigned to its closest representative with the use of a similarity or distance function. Therefore, the data partitioning can be uniquely defined by the set of representatives. For an $m \times n$ data set, each representative $\overline{Y_i}$ is an n-dimensional data point, which is a central point of the ith cluster. Ideally, we would like the central representative to be the mean of the cluster.

Therefore, the clusters are dependent on the representatives and vice versa. Such an interdependency is achieved with an iterative approach. We start with a set of representatives $\overline{Y_1} \ldots \overline{Y_k}$, which might be randomly chosen points generated in the range of the data space. We iteratively compute the cluster partitions using the representatives, and then recompute the representatives as the centroids of the resulting clusters. While computing the centroids, care must be taken to use only the observed values in each dimension. This two-step iterative approach is executed to convergence. The two-step approach is summarized as follows:

1. Determine the clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ by assigning each row in the $m \times n$ matrix to its closest representative from $\overline{Y_1} \ldots \overline{Y_k}$. Typically, the Euclidean distance or the Manhattan distance is used for similarity computation.

2. For each $i \in \{1 \ldots k\}$, reset $\overline{Y_i}$ to the centroid of the current set of points in $\mathcal{C}_i$.

The main problem with the use of this approach is that the $m \times n$ ratings matrix is incomplete. Therefore, the computation of the mean and the distance values becomes undefined. However, it is relatively easy to compute the means using only the observed values within a cluster. In some cases, the centroid itself might not be fully specified, when no rating is specified for one or more items in the cluster. The distance values are computed using only the subset of dimensions, which are specified both for the data point and cluster representative. The distance is also divided by the number of dimensions used in the computation. This is done in order to adjust for the fact that different numbers of dimensions are used for computing the distance of a data point to various centroids, when all the centroids are not fully specified. In this context, the Manhattan distance yields better adjustments than the Euclidean distance, and the normalized value can be interpreted more easily as an average distance along each observed value.

The aforementioned approach clusters the rows for user-based collaborative filtering. In item-based methods, it would be necessary to cluster the columns. The approach is exactly similar except that it is applied to the columns rather than the rows. A number of clustering methods for efficient collaborative filtering are discussed in [146, 167, 528, 643,

644, 647]. Some of these methods are user-based methods, whereas others are item-based methods. A number of co-clustering methods [643] can be used to cluster rows and columns simultaneously.

2.5 Dimensionality Reduction and Neighborhood Methods

Dimensionality reduction methods can be used to improve neighborhood-based methods both in terms of quality and in terms of efficiency. In particular, even though pairwise similarities are hard to robustly compute in sparse rating matrices, dimensionality reduction provides a dense low-dimensional representation in terms of latent factors. Therefore, such models are also referred to as *latent factor models*. Even when two users have very few items rated in common, a distance can be computed between their low-dimensional latent vectors. Furthermore, it is more efficient to determine the peer groups with low-dimensional latent vectors. Before discussing the details of dimensionality reduction methods, we make some comments about two distinct ways in which latent factor models are used in recommender systems:

1. A reduced representation of the data can be created in terms of *either* row-wise latent factors or in terms of column-wise latent factors. In other words, the reduced representation will either compress the item dimensionality or the user dimensionality into latent factors. This reduced representation can be used to alleviate the sparsity problem for neighborhood-based models. Depending on which dimension has been compressed into latent factors, the reduced representation can be used for either user-based neighborhood algorithms or item-based neighborhood algorithms.

2. The latent representations of *both* the row space and the column space are determined simultaneously. These latent representations are used to reconstruct the entire ratings matrix in one shot without the use of neighborhood-based methods.

Because the second class of methods is not directly related to neighborhood-based methods, it will not be discussed in this chapter. A detailed discussion of the second class of methods will be provided in Chapter 3. In this chapter, we will focus only on the first class of methods.

For ease of discussion, we will first describe only the user-based collaborative filtering method. In user-based collaborative filtering methods, the basic idea is to transform the $m \times n$ ratings matrix R into a lower-dimensional space by using principal component analysis. The resulting matrix R' is of size $m \times d$, where $d \ll n$. Thus, each of the (sparse) n-dimensional vector of ratings corresponding to a user is transformed into a reduced d-dimensional space. Furthermore, unlike the original rating vector, each of the d dimensions is fully specified. After this d-dimensional representation of each user is determined, the similarity is computed from the target user to each user using the reduced representation. The similarity computations in the reduced representation are more robust because the new low-dimensional vector is fully specified. Furthermore, the similarity computations are more efficient because of the low dimensionality of the latent representation. A simple cosine or dot product on the reduced vectors is sufficient to compute the similarity in this reduced space.

It remains to be described how the low-dimensional representation of each data point is computed. The low-dimensional representation can be computed using either SVD-like methods or PCA-like methods. In the following, we describe an SVD-like method.

Table 2.3: Example of bias in estimating covariances

User Index	*Godfather*	*Gladiator*	*Nero*
1	1	1	1
2	7	7	7
3	3	1	1
4	5	7	7
5	3	1	?
6	5	7	?
7	3	1	?
8	5	7	?
9	3	1	?
10	5	7	?
11	3	1	?
12	5	7	?

The first step is to augment the $m \times n$ incomplete ratings matrix R to fill in the missing entries. The missing entry is estimated to be equal to the mean of the corresponding row in the matrix (i.e., the mean rating of the corresponding user). An alternative approach is to estimate the missing entry as the mean of the corresponding column in the matrix (i.e., the mean rating of the corresponding item). Let the resulting matrix be denoted by R_f. Then, we compute the $n \times n$ similarity matrix between pairs of items, which is given by $S = R_f^T R_f$. This matrix is positive semi-definite. In order to determine the dominant basis vectors of R_f for SVD, we perform the diagonalization of the similarity matrix S as follows:

$$S = P \Delta P^T \tag{2.16}$$

Here, P is an $n \times n$ matrix, whose columns contain the orthonormal eigenvectors of S. Δ is a diagonal matrix containing the non-negative eigenvalues of S along its diagonal. Let P_d be the $n \times d$ matrix containing only the columns of P corresponding to the largest d eigenvectors. Then, the low-dimensional representation of R_f is given by the matrix product $R_f P_d$. Note that the dimensions of the reduced representation $R_f P_d$ are $m \times d$, because R_f is an $m \times n$ matrix and P_d is an $n \times d$ matrix. Therefore, each of the m users is now represented in a d-dimensional space. This representation is then used to determine the peer group of each user. Once the peers have been determined, the rating prediction can be easily performed with Equation 2.4. Such an approach can also be used for item-based collaborative filtering by applying the entire dimensionality reduction method to the transpose of R_f instead of R_f.

The aforementioned methodology can be viewed as a *singular value decomposition (SVD)* of the ratings matrix R_f. A number of other methods [24, 472] use *principal component analysis (PCA)* instead of SVD, but the overall result is very similar. In the PCA method, the covariance matrix of R_f is used instead of the similarity matrix $R_f^T R_f$. For data, which is mean-centered along columns, the two methods are identical. Therefore, one can subtract the mean of each column from its entries, and then apply the aforementioned approach to obtain a transformed representation of the data. This transformed representation is used to determine the peers of each user. Mean-centering has benefits in terms of reducing *bias* (see next section). An alternative approach is to first mean center along each row and then mean-center along each column. SVD can be applied to the transformed representation. This type of approach generally provides the most robust results.

2.5.1 Handling Problems with Bias

It is noteworthy that the matrix R_f is derived from the incomplete matrix R by filling in the unspecified entries with average values along either the rows or the columns. Such an approach is likely to cause considerable *bias*. To understand the nature of this bias, consider the example in Table 2.3 of ratings given by 12 users to the three movies *Godfather*, *Gladiator*, and *Nero*. Let us assume that PCA is used for dimensionality reduction, and therefore the covariance matrix needs to be estimated. Let us assume that missing values are replaced with the averages along the columns.

In this case, the ratings are drawn on a scale from 1 to 7 by a set of 4 users for 3 movies. It is visually evident that the correlations between the ratings of the movies *Gladiator* and *Nero* are extremely high because the ratings are very similar in the four cases in which they are specified. The correlation between *Godfather* and *Gladiator* seems to be less significant. However, many users have not specified their ratings for *Nero*. Because the mean rating of *Nero* is $(1 + 7 + 1 + 7)/4 = 4$, these unspecified ratings are replaced with the mean value of 4. The addition of these new entries significantly reduces the estimated covariance between *Gladiator* and *Nero*. However, the addition of the new entries has no impact on the covariance between *Godfather* and *Gladiator*. After filling in the missing ratings, the pairwise covariances between the three movies can be estimated as follows:

	Godfather	*Gladiator*	*Nero*
Godfather	2.55	4.36	2.18
Gladiator	4.36	9.82	3.27
Nero	2.18	3.27	3.27

According to the aforementioned estimation, the covariance between *Godfather* and *Gladiator* is larger than that between *Gladiator* and *Nero*. This does not seem to be correct because the ratings in Table 2.3 for *Gladiator* and *Nero* are identical for the case where both are specified. Therefore, the correlation between *Gladiator* and *Nero* ought to be higher. This error is a result of the bias caused by filling in the unspecified entries with the mean of that column. This kind of bias can be very significant in sparse matrices because most of the entries are unspecified. Therefore, methods need to be designed to reduce the bias caused by using the mean ratings in place of the unspecified entries. In the following, we explore two possible solutions to this problem.

2.5.1.1 Maximum Likelihood Estimation

The conceptual reconstruction method [24, 472] proposes the use of probabilistic techniques, such as the EM-algorithm, in order to estimate the covariance matrix. A generative model is assumed for the data and the specified entries are viewed as the outcomes of the generative model. The covariance matrix can be estimated as part of the process of estimating the parameters of this generative model. In the following, we provide a simplification of this approach. In this simplified approach, the maximum likelihood estimate of the covariance matrix is computed. The maximum likelihood estimate of the covariance between each pair of items is estimated as the covariance between only the specified entries. In other words, only the users that have specified ratings for a particular pair of items are used to estimate the covariance. In the event that there are no users in common between a pair of items, the covariance is estimated to be 0. By using this approach, the following covariance matrix is estimated for the data in Table 2.3.

	Godfather	*Gladiator*	*Nero*
Godfather	2.55	4.36	8
Gladiator	4.36	9.82	12
Nero	8	12	12

In this case, it becomes immediately evident that the covariance between *Gladiator* and *Nero* is almost three times that between *Godfather* and *Gladiator*. Furthermore, the movie *Nero* has more than three times as much variance than was originally estimated and has the largest variance in ratings among all movies. While the pairwise covariance between *Godfather* and *Gladiator* was the largest compared to all other pairwise covariances using the mean-filling technique, this same pair now shows the least of all pairwise covariances. This example suggests that the bias corrections can be very significant in some situations. The greater the proportion of unspecified entries in the matrix, the greater the bias of the mean-filling technique. Therefore, the modified technique of leveraging only the specified entries is used for computing the covariance matrix. While such a technique is not always effective, it is superior to the mean-filling technique. The reduced $n \times d$ basis matrix P_d is computed by selecting the top-d eigenvectors of the resulting covariance matrix.

In order to further reduce the bias in representation, the incomplete matrix R can be directly projected on the reduced matrix P_d, rather than projecting the filled matrix R_f on P_d. The idea is to compute the contribution of each observed rating to the projection on each latent vector of P_d, and then average the contribution over the number of such ratings. This averaged contribution is computed as follows. Let $\overline{e_i}$ be the ith column (eigenvector) of P_d, for which the jth entry is e_{ji}. Let r_{uj} be the observed rating of user u for item j in matrix R. Then, the contribution of user u to the projection on latent vector $\overline{e_i}$ is given by $r_{uj}e_{ji}$. Then, if the set I_u represents the indices of the specified item ratings of user u, the averaged contribution a_{ui} of user u on the ith latent vector is as follows:

$$a_{ui} = \frac{\sum_{j \in I_u} r_{uj} e_{ji}}{|I_u|} \tag{2.17}$$

This type of averaged normalization is particularly useful in cases where the different users have specified different numbers of ratings. The resulting $m \times d$ matrix $A = [a_{ui}]_{m \times d}$ is used as the reduced representation of the underlying ratings matrix. This reduced matrix is used to compute the neighborhood of the target user efficiently for user-based collaborative filtering. It is also possible to apply the approach to the transpose of the matrix R and reduce the dimensionality along the user dimension, rather than the item dimension. Such an approach is useful for computing the neighborhood of a target item in item-based collaborative filtering. This approach of using the reduced representation for missing value imputation is discussed in [24, 472].

2.5.1.2 Direct Matrix Factorization of Incomplete Data

Although the aforementioned methodology can correct for the bias in covariance estimation to some extent, it is not completely effective when the sparsity level of the ratings is high. This is because the covariance matrix estimation requires a sufficient number of observed ratings for each pair of items for robust estimation. When the matrix is sparse, the covariance estimates will be statistically unreliable.

A more direct approach is to use matrix factorization methods. Methods such as singular value decomposition are essentially matrix factorization methods. For a moment, assume

that the $m \times n$ ratings matrix R is fully specified. It is a well-known fact of linear algebra [568] that any (fully specified) matrix R can be factorized as follows:

$$R = Q \Sigma P^T \tag{2.18}$$

Here, Q is an $m \times m$ matrix with columns containing the m orthonormal eigenvectors of RR^T. The matrix P is an $n \times n$ matrix with columns containing the n orthonormal eigenvectors of $R^T R$. Σ is an $m \times n$ diagonal matrix in which only diagonal entries[4] are nonzero and they contain the square-root of the nonzero eigenvalues of $R^T R$ (or equivalently, RR^T). It is noteworthy that the eigenvectors of $R^T R$ and RR^T are not the same and will have different dimensionality when $m \neq n$. However, they will always have the same number of (nonzero) eigenvalues, which are identical in value. The values on the diagonal of Σ are also referred to as *singular values*.

Furthermore, one can *approximately* factorize the matrix by using *truncated* SVD, where only the eigenvectors corresponding to the $d \leq \min\{m, n\}$ largest singular values are used. Truncated SVD is computed as follows:

$$R \approx Q_d \Sigma_d P_d^T \tag{2.19}$$

Here, Q_d, Σ_d, and P_d are $m \times d$, $d \times d$, and $n \times d$ matrices, respectively. The matrices Q_d and P_d, respectively, contain the d largest eigenvectors of RR^T and $R^T R$, whereas the matrix Σ_d contains the square-roots of the d largest eigenvalues of either matrix along its diagonal. It is noteworthy that the matrix P_d contains the top eigenvectors of $R^T R$, which is the *reduced* basis representation required for dimensionality reduction. Furthermore, the matrix $Q_d \Sigma_d$ contains the transformed and reduced $m \times d$ representation of the original ratings matrix in the basis corresponding to P_d. It can be shown that such an approximate factorization has the least mean-squared error of the approximated entries as compared to any other rank-d factorization. Therefore, if we can approximately factorize the ratings matrix R in the form corresponding to Equation 2.19, it provides us with the reduced basis as well as the representation of the ratings in the reduced basis. The main problem of using such an approach is that the ratings matrix is not fully specified. As a result, this factorization is undefined. Nevertheless, it is possible to recast the formulation as an optimization problem, in which the squared error of factorization is optimized *only over the observed entries* of the ratings matrix. It is also possible to explicitly solve this modified formulation using nonlinear optimization techniques. This results in a robust and unbiased lower dimensional representation. Furthermore, such an approach can be used to directly estimate the ratings matrix by using Equation 2.19, once the reduced factor matrices have been determined. In other words, such methods have a direct utility beyond neighborhood-based methods. More details of these latent factor models and nonlinear optimization techniques will be discussed in section 3.6 of Chapter 3. The reader should consult this section to learn how the reduced representation may be computed by using modified optimization formulations.

2.6 A Regression Modeling View of Neighborhood Methods

An important observation about both user-based and item-based methods is that they predict ratings as *linear functions* of either the ratings of the same item by neighboring users, or of the same user on neighboring items. In order to understand this point, we

[4] Diagonal matrices are usually square. Although this matrix is not square, only entries with equal indices are nonzero. This is a generalized definition of a diagonal matrix.

replicate the prediction function of user-based neighborhood methods (cf. Equation 2.4) below:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.20}$$

Note that the predicted rating is a *weighted* linear combination of other ratings of the same item. The linear combination has been restricted only to the ratings of item j belonging to users with sufficiently similar tastes to target user u. This restriction is enabled with the use of the peer rating set $P_u(j)$. Recall from the discussion earlier in this chapter that $P_u(j)$ is the set of k nearest users to target user u, who have also rated item j. Note that if we allowed the set $P_u(j)$ to contain all ratings of item j (and not just specific peer users), then the prediction function becomes similar[5] to that of linear regression [22]. In linear regression, the ratings are also predicted as weighted combinations of other ratings, and the weights (coefficients) are determined with the use of an optimization model. In the neighborhood-based approach, the coefficients of the linear function are chosen in a heuristic way with the user-user similarities, rather than with the use of an optimization model.

A similar observation applies to the case of item-based neighborhood methods, where the prediction function (cf. Equation 2.15) is as follows:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j, t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j, t)|} \tag{2.21}$$

The set $Q_t(u)$ represents the set of the k closest items to target item t that have also been rated by user u. In this case, the rating of a user u for a target item t is expressed as a linear combination of her *own* ratings. As in the case of user-based methods, the coefficients of the linear combination are heuristically defined with similarity values. Therefore, a user-based model expresses a predicted rating as a linear combination of ratings in the same *column*, whereas an item-based model expresses a predicted rating as a linear combination of ratings in the same *row*. From this point of view, *neighborhood-based models are heuristic variants of linear regression models*, in which the regression coefficients are heuristically set to similarity values for related (neighboring) items/users and to 0 for unrelated items/users.

It is noteworthy that the use of similarity values as combination weights is rather heuristic and arbitrary. Furthermore, the coefficients do not account for interdependencies among items. For example, if a user has rated certain sets of correlated items in a very similar way, then the coefficients associated with these items will be interdependent as well. The use of similarities as heuristic weights does not account for such interdependencies.

A question arises as to whether one can do better by *learning* the weights with the use of an optimization formulation. It turns out that one can derive analogous regression-based models to the user-based and item-based models. Several different optimization formulations have been proposed in the literature, which can leverage user-based models, item-based models, or a combination of the two. These models can be viewed as theoretical generalizations of the heuristic nearest neighbor model. The advantage of such models is that they are mathematically better founded in the context of a crisp optimization formulation, and the weights for combining the ratings can be better justified because of their optimality from a *modeling* perspective. In the following, we discuss an optimization-based neighborhood model, which is a simplification of the work in [309]. This also sets the stage for combining the power of this model with other optimization models, such as matrix factorization, in section 3.7 of Chapter 3.

[5] A discussion of linear regression is provided in section 4.4.5 of Chapter 4, but in the context of content-based systems.

2.6.1 User-Based Nearest Neighbor Regression

Consider the user-based prediction of Equation 2.20. One can replace the (normalized) similarity coefficient with the unknown parameter w_{vu}^{user} to *model* the predicted rating $\hat{r}_{uj}$ of target user u for item j as follows:

$$\hat{r}_{uj} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.22}$$

As in the case of neighborhood models, one can use the Pearson correlation coefficient to define $P_u(j)$. There is, however, a subtle but important difference in terms of how $P_u(j)$ is defined in this case. In neighborhood-based models, $P_u(j)$ is the set of k closest users to target user u, who have specified ratings for item j. Therefore, the size of $P_u(j)$ is often exactly k, when at least k users have rated item j. In the case of regression methods, the set $P_u(j)$ is defined by first determining the k closest peers for each user, and then retaining only those for which ratings are observed. Therefore, the size of set $P_u(j)$ is often *significantly* less than k. Note that the parameter k needs to be set to much larger values in the regression framework as compared to that in neighborhood models because of its different interpretation.

Intuitively, the unknown coefficient w_{vu}^{user} controls the portion of the prediction of ratings given by user u, which comes from her similarity to user v, because this portion is given by $w_{vu}^{user} \cdot (r_{vj} - \mu_v)$. It is possible for w_{vu}^{user} to be different from w_{uv}^{user}. It is also noteworthy that w_{vu}^{user} is only defined for the k different values of v (user indices) that are closest to user u on the basis of the Pearson coefficient. The other values of w_{vu}^{user} are not needed by the prediction function of Equation 2.22, and they therefore do not need to be learned. This has the beneficial effect of reducing the number of regression coefficients.

One can use the aggregate squared difference between the predicted ratings $\hat{r}_{uj}$ (according to Equation 2.22) and the observed ratings r_{uj} to create an objective function that estimates the quality of a particular set of coefficients. Therefore, one can use the observed ratings in the matrix to set up a least-squares optimization problem over the unknown values of w_{vu}^{user} in order to minimize the overall error. The idea is to predict each (observed) rating of user u with her nearest k users in a formal regression model, and then measure the error of the prediction. The squared errors can be added over all items rated by user u to create a least-squares formulation. Therefore, the optimization problem is set up for each target user u. Let I_u be the set of items that have been rated by the target user u. The least-squares objective function for the uth user can be stated as the sum of the squares of the errors in predicting each item in I_u with the k nearest neighbors of the user in a formal regression model:

$$\begin{aligned} \text{Minimize } J_u &= \sum_{j \in I_u} (r_{uj} - \hat{r}_{uj})^2 \\ &= \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2 \end{aligned}$$

The second relationship is obtained by substituting the expression in Equation 2.22 for $\hat{r}_{uj}$. Note that this optimization problem is formulated separately for each target user u. However, one can add up the objective function values J_u over different target users $u \in \{1 \ldots m\}$ with no difference to the optimal solution. This is because the various values of J_u are expressed in terms of mutually disjoint sets of optimization variables w_{vu}^{user}. Therefore, the

consolidated optimization problem is expressed as follows:

$$\text{Minimize} \sum_{u=1}^{m} J_u = \sum_{u=1}^{m} \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2 \tag{2.23}$$

One can solve each of the smaller optimization problems (i.e., objective function J_u) in their decomposed form more efficiently without affecting the overall solution. However, the consolidated formulation has the advantage that it can be combined with other optimization models such as matrix factorization methods (cf. section 3.7 of Chapter 3) in which such a decomposition is not possible. Nevertheless, if linear regression is to be used on a standalone basis, it makes sense to solve these problems in their decomposed form.

Both the consolidated and decomposed versions of the optimization models are least-squares optimization problems. These methods can be solved with the use of any off-the-shelf optimization solver. Refer to section 4.4.5 of Chapter 4 for a discussion of closed form solutions to linear regression problems. A desirable property of most of these solvers is that they usually have *regularization* built in them, and they can therefore avoid overfitting to some extent. The basic idea in regularization is to reduce model complexity by adding the term $\lambda \sum_{j \in I_u} \sum_{v \in P_u(j)} (w_{vu}^{user})^2$ to each (decomposed) objective function J_u, where $\lambda > 0$ is a user-defined parameter regulating the weight of the regularization term. The term $\lambda \sum_{j \in I_u} \sum_{v \in P_u(j)} (w_{vu}^{user})^2$ penalizes large coefficients, and it therefore shrinks the absolute values of the coefficients. Smaller coefficients result in simpler models and reduce overfitting. However, as discussed below, it is sometimes not sufficient to use regularization alone to reduce overfitting.

2.6.1.1 Sparsity and Bias Issues

One problem with this regression approach is that the size of the $P_u(j)$ can be vastly different for the same user u and varying item indices (denoted by j). This is because of the extraordinary level of sparsity inherent in ratings matrices. As a result, the regression coefficients become heavily dependent on the *number* of peer users that have rated a particular item j along with user u. For example, consider a scenario where the target user u has rated both *Gladiator* and *Nero*. Out of the k nearest neighbors of the target u, only one user might rate the movie *Gladiator*, whereas all k might have rated *Nero*. As a result, the regression coefficient w_{vu}^{user} of the peer user v who rated *Gladiator* will be heavily influenced by the fact that she is the only user who has rated *Gladiator*. This will result in overfitting because this (statistically unreliable) regression coefficient might add noise to the rating predictions of other movies.

The basic idea is to change the prediction function and assume that the regression for item j predicts only a fraction $\frac{|P_u(j)|}{k}$ of the rating of target user u for item j. The implicit assumption is that the regression coefficients are based on *all* the peers of the target user, and one must interpolate incomplete information as a fraction. Therefore, this approach changes the interpretation of the regression coefficients. In this case, the prediction function of Equation 2.22 is modified as follows:

$$\hat{r}_{uj} \cdot \frac{|P_u(j)|}{k} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.24}$$

A number of other heuristic adjustments are sometimes used. For example, along the lines of the ideas in [312], one can use a heuristic adjustment factor of $\sqrt{|P_u(j)|/k}$. This factor can

often be simplified to $\sqrt{|P_u(j)|}$ because constant factors are absorbed by the optimization variables. A related enhancement is that the *constant* offset μ_v is replaced with a bias *variable* b_u, which is learned in the optimization process. The corresponding prediction model, including heuristic adjustment factors, is as follows:

$$\hat{r}_{uj} = b_u^{user} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - b_v^{user})}{\sqrt{|P_u(j)|}} \tag{2.25}$$

Note that this model is no longer linear because of the multiplicative term $w_{vu}^{user} \cdot b_v^{user}$ between two optimization variables. Nevertheless, it is relatively easy to use the same least-squares formulation, as in the previous case. In addition to user biases, one can also incorporate *item* biases. In such a case, the model becomes the following:

$$\hat{r}_{uj} = b_u^{user} + b_j^{item} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - b_v^{user} - b_j^{item})}{\sqrt{|P_u(j)|}} \tag{2.26}$$

Furthermore, it is recommended to center the entire ratings matrix around its global mean by subtracting the mean of all the observed entries from it. The global mean needs to be added back to the predictions. The main problem with this model is computational. One must pre-compute and store all user-user relations, which is computationally expensive and requires $O(m^2)$ space over m users. This problem is similar to that encountered in traditional neighborhood-based models. Such models are suitable in settings in which the item space changes rapidly, but the users are relatively stable over time [312]. An example is the case of news recommender systems.

2.6.2 Item-Based Nearest Neighbor Regression

The item-based approach is similar to the user-based approach, except that the regression learns and leverages item-item correlations rather than user-user correlations. Consider the item-based prediction of Equation 2.21. One can replace the (normalized) similarity coefficient AdjustedCosine(j, t) with the unknown parameter w_{jt}^{item} to model the rating prediction of user u for target item t:

$$\hat{r}_{ut} = \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj} \tag{2.27}$$

The nearest items in $Q_t(u)$ can be determined using the adjusted cosine, as in item-based neighborhood methods. The set $Q_t(u)$ represents the subset of the k nearest neighbors of the target item t, for which user u has provided ratings. This way of defining $Q_t(u)$ is subtly different from that of traditional neighborhood-based methods, because the size of set $Q_t(u)$ might be significantly less than k. In traditional neighborhood methods, one determines the closest k items to target item t, for which the user u has specified ratings, and therefore the size of the neighborhood set is often exactly k. This change is required to be able to effectively implement the regression-based method.

Intuitively, the unknown coefficient w_{jt}^{item} controls the portion of the rating of item t, which comes from its similarity to item j, because this portion is given by $w_{jt}^{item} \cdot r_{uj}$. The prediction error of Equation 2.27 should be minimized to ensure the most robust predictive model. One can use the known ratings in the matrix to set up a least-squares optimization problem over the unknown values of w_{jt}^{item} in order to minimize the overall error. The idea is to predict each (observed) rating of target item t with its nearest k items and then,

create an expression for the least-squares error. The optimization problem is set up for each target item t. Let U_t be the set of users who have rated the target item t. The least-squares objective function for the tth item can be stated as the sum of the squares of the errors in predicting each specified rating in U_t:

$$\begin{aligned}\text{Minimize } J_t &= \sum_{u \in U_t} (r_{ut} - \hat{r}_{ut})^2 \\ &= \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2\end{aligned}$$

Note that this optimization problem is formulated separately for each target item t. However, one can add up the terms over various values of the target item t with no difference to the optimization solution, because the unknown coefficients w_{jt}^{item} in the various objective functions are non-overlapping over different values of the target item $t \in \{1 \ldots n\}$. Therefore, we have the following consolidated formulation:

$$\text{Minimize } \sum_{t=1}^{n} \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2 \tag{2.28}$$

This is a least-squares regression problem and it can be solved with the use of any off-the-shelf solver. Furthermore, one can also solve each of the smaller optimization problems (i.e., objective function J_t) in its decomposed form more efficiently without affecting the overall solution. However, the consolidated formulation has the advantage that it can be combined with other optimization models, such as matrix factorization methods (cf. section 3.7 of Chapter 3). As in the case of user-based methods, significant challenges are associated with the problem of overfitting. One can add the regularization term $\lambda \sum_{u \in U_t} \sum_{j \in Q_t(u)} (w_{jt}^{item})^2$ to the objective function J_t.

As discussed in section 2.6.1.1 for the case of the user-based model, one can incorporate adjustment factors and bias variables to improve performance. For example, the user-based prediction model of Equation 2.26 takes on the following form in the item-wise model:

$$\hat{r}_{ut} = b_u^{user} + b_t^{item} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - b_u^{user} - b_j^{item})}{\sqrt{|Q_t(u)|}} \tag{2.29}$$

Furthermore, it is assumed that the ratings are centered around the global mean of the entire ratings matrix. Therefore, the global mean is subtracted from each of the ratings before building the model. All predictions are performed on the centered ratings, and then the global mean is added back to each prediction. In some variations of the model, the bias terms $b_u^{user} + b_j^{item}$ within brackets are replaced with a consolidated *constant* term B_{uj}. This constant term is derived using a non-personalized approach described in section 3.7.1 of Chapter 3. The resulting prediction model is as follows:

$$\hat{r}_{ut} = b_u^{user} + b_t^{item} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - B_{uj})}{\sqrt{|Q_t(u)|}} \tag{2.30}$$

A least-squares optimization model is formulated, and a gradient descent approach is used to solve for the optimization parameters. This is precisely the model used in [309]. The resulting gradient-descent steps are discussed in section 3.7.2 of Chapter 3. The user-user model is known to perform slightly better than the item-item model [312]. However, the item-based model is far more computationally and space-efficient in settings where the number of items is much smaller than the number of users.

2.6.3 Combining User-Based and Item-Based Methods

It is natural to combine the user and item-based models in a unified regression framework [312]. Therefore, a rating is predicted based on its relationship with similar users as well as similar items. This is achieved by combining the ideas in Equations 2.26 and 2.30 as follows:

$$\hat{r}_{uj} = b_u^{user} + b_j^{item} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - B_{vj})}{\sqrt{|P_u(j)|}} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - B_{uj})}{\sqrt{|Q_t(u)|}} \quad (2.31)$$

As in previous cases, it is assumed that the ratings matrix is centered around its global mean. A similar least-squares optimization formulation can be used in which the squared error over all the observed entries is minimized. In this case, it is no longer possible to decompose the optimization problem into independent subproblems. Therefore, a single least-squares optimization model is constructed over all the observed entries in the ratings matrix. As in the previous cases, the gradient-descent approach can be used. It was reported in [312] that the fusion of the user-based and item-based models generally performs better than the individual models.

2.6.4 Joint Interpolation with Similarity Weighting

The method in [72] uses a different idea to set up the joint neighborhood-based model. The basic idea is to predict each rating of target user u with the user-based model of Equation 2.22. Then, instead of comparing it with the observed value of the *same* item, we compare it with the observed ratings of *other* items of that user.

Let S be the set of all pairs of user-item combinations in the ratings matrix, which have been observed:

$$S = \{(u, t) : r_{ut} \text{ is observed}\} \quad (2.32)$$

We set up an objective function which is penalized when the predicted rating $\hat{r}_{uj}$ of an item j is far away from the observed rating given to a similar item s by the same target user u. In other words, the objective function for target user u is defined as follows:

$$\text{Minimize} \sum_{s:(u,s) \in S} \sum_{j:j \neq s} AdjustedCosine(j, s) \cdot (r_{us} - \hat{r}_{uj})^2$$

$$= \sum_{s:(u,s) \in S} \sum_{j:j \neq s} AdjustedCosine(j, s) \cdot \left(r_{us} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2$$

Regularization can be added to the objective function to reduce overfitting. Here, $P_u(j)$ is defined as the k closest users to target user u, who have also rated item j. Therefore, the conventional definition of $P_u(j)$ as used in neighborhood-based models is leveraged in this case.

By using the adjusted cosine as a multiplicative factor of each individual term in the objective function, the approach forces the target user's ratings of similar items to be more similar as well. It is noteworthy that both user and item similarities are used in this approach, but in different ways:

1. The item-item similarities are used as multiplicative factors of the terms in the objective function to force predicted ratings to be more similar to observed ratings of similar items.

2. The user-user similarities are used for predicting the ratings by restricting the regression coefficients to the relevant peer group $P_u(j)$ of the target user u.

Although it is also possible, in principle, to switch the roles of users and items to set up a different model, it is stated in [72] that the resulting model is not as effective as the one discussed above. This model can be solved with any off-the-shelf least-squares solver. A number of methods are also discussed in [72] for handling sparsity.

2.6.5 Sparse Linear Models (SLIM)

An interesting method, based on the item-item regression in section 2.6.2, is proposed in [455]. This family of models is referred to as *sparse linear models* because they encourage sparsity in the regression coefficients with the use of regularization methods. Unlike the methods in [72, 309], these methods work with non-negative rating values. Therefore, unlike the techniques in the previous sections, it will not be assumed that the ratings matrix is mean-centered. This is because mean-centering will automatically create negative ratings, corresponding to dislikes. However, the approach is designed to work with non-negative ratings, in which there is no mechanism to specify dislikes. From a practical point of view, the approach is most appropriate[6] for implicit feedback matrices (e.g., click-through data or sales data), where only positive preferences are expressed through user actions. Furthermore, as is common in implicit feedback settings, missing values are treated as 0s for the purposes of training in the optimization formulation. However, the optimization model might eventually predict some of these values to be highly positive, and such user-item combinations are excellent candidates for recommendation. Therefore, the approach ranks items on the basis of prediction errors on the training entries that have been set to 0.

Unlike the technique in section 2.6.2, these methods do not restrict the regression coefficients to only the neighborhood of the target item t. Then, the prediction function in *SLIM* is expressed as follows:

$$\hat{r}_{ut} = \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj} \quad \forall u \in \{1 \ldots m\}, \ \forall t \in \{1 \ldots n\} \tag{2.33}$$

Note the relationship with Equation 2.27 in which only the neighborhood of the target item is used to construct the regression. It is important to exclude the target item itself on the right-hand side to prevent overfitting. This can be achieved by requiring the constraint that $w_{tt}^{item} = 0$. Let $\hat{R} = [\hat{r}_{uj}]$ represent the predicted ratings matrix and let $W^{item} = [w_{jt}^{item}]$ represent the item-item regression matrix. Therefore, if we assume that the diagonal elements of W^{item} are constrained to be 0, then we can stack up the instantiations of Equation 2.33 over different users and target items to create the following matrix-based prediction function:

$$\hat{R} = RW^{item}$$
$$\text{Diagonal}(W^{item}) = 0$$

Therefore, the main goal is to minimize the Frobenius norm $||R - RW^{item}||^2$ along with some regularization terms. This objective function is disjoint over different columns of W (i.e., target items in regression). Therefore, one can solve each optimization problem (for

[6]The approach can be adapted to arbitrary rating matrices. However, the main advantages of the approach are realized for non-negative ratings matrices.

a given value of the target item t) independently, while setting w_{tt}^{item} to 0. In order to create a more interpretable sum-of-parts regression, the weight vectors are constrained to be non-negative. Therefore, the objective function for target item t may be expressed as follows:

$$\text{Minimize } J_t^s = \sum_{u=1}^{m} (r_{ut} - \hat{r}_{ut})^2 + \lambda \cdot \sum_{j=1}^{n} (w_{jt}^{item})^2 + \lambda_1 \cdot \sum_{j=1}^{n} |w_{jt}^{item}|$$

$$= \sum_{u=1}^{m} (r_{ut} - \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj})^2 + \lambda \cdot \sum_{j=1}^{n} (w_{jt}^{item})^2 + \lambda_1 \cdot \sum_{j=1}^{n} |w_{jt}^{item}|$$

$$\text{subject to:}$$

$$w_{jt}^{item} \geq 0 \quad \forall j \in \{1 \ldots n\}$$

$$w_{tt}^{item} = 0$$

The last two terms in the objective function correspond to the *elastic-net regularizer*, which combines L_1- and L_2-regularization. It can be shown [242] that the L_1-regularization component leads to sparse solutions for the weights w_{jt}, which means that most of the coefficients w_{jt} have zero values. The sparsity ensures that each predicted rating can be expressed as a more interpretable linear combination of the ratings of a small number of other related items. Furthermore, since the weights are non-negative, the corresponding items are positively related in a highly interpretable way in terms of the specific level of impact of each rating in the regression. The optimization problem is solved using the coordinate descent method, although any off-the-shelf solver can be used in principle. A number of faster techniques are discussed in [347]. The technique can also be hybridized [456] with side-information (cf. section 6.8.1 of Chapter 6).

It is evident that this model is closely related to the neighborhood-based regression models discussed in the previous sections. The main differences of the *SLIM* model from the linear regression model in [309] are as follows:

1. The method in [309] restricts the nonzero coefficients for each target to at most the k most similar items. The *SLIM* method can use as many as $|U_t|$ nonzero coefficients. For example, if an item is rated by all users, then all coefficients will be used. However, the value of w_{tt}^{item} is set to 0 to avoid overfitting. Furthermore, the *SLIM* method forces sparsity by using the elastic-net regularizer, whereas the method in [309] preselects the weights on the basis of explicit neighborhood computation. In other words, the work in [309] uses a heuristic approach for feature selection, whereas the *SLIM* approach uses a learning (regularization) approach for feature selection.

2. The *SLIM* method is primarily designed for implicit feedback data sets (e.g., buying an item or customer clicks), rather than explicit ratings. In such cases, ratings are typically unary, in which customer actions are indications of positive preference, but the act of not buying or clicking on an item does not necessarily indicate a negative preference. The approach can also be used for cases in which the "ratings" are arbitrary values indicating only positive preferences (e.g., amount of product bought). Note that such scenarios are generally conducive to regression methods that impose non-negativity in the coefficients of the model. As you will learn in Chapter 3, this observation is also true for other models, such as matrix factorization. For example, non-negative matrix factorization is primarily useful for implicit feedback data sets, but it is not quite as useful for arbitrary ratings. This is, in part, because the non-negative, sum-of-parts decomposition loses its interpretability when a rating indicates either a like or a dislike. For example, two "dislike" ratings do not add up to a "like" rating.

3. The regression coefficients in [309] can be either positive or negative. On the other hand, the coefficients in *SLIM* are constrained to be non-negative. This is because the *SLIM* method is primarily designed for the implicit feedback setting. Non-negativity is often more intuitive in these settings and the results are more interpretable. In fact, in some cases, imposing non-negativity might improve[7] the accuracy. However, some limited experimental results have been presented [347], which suggest that removing non-negativity constraints provides superior performance.

4. Although the *SLIM* method also proposes a prediction model for the ratings (according to Equation 2.33), the final *use* of the predicted values is for *ranking* the items in order of the predicted value. Note that the approach is generally used for data sets with unary ratings and therefore, it makes sense to use the predicted values to rank the items, rather than predict ratings. An alternative way of interpreting the predicted values is that each of them can be viewed as the *error* of replacing a non-negative rating with 0 in the ratings matrix. The larger the error is, the greater the predicted value of the rating will be. Therefore, the items can be ranked in the order of the predicted value.

5. Unlike the work in [309], the *SLIM* method does not explicitly adjust for the varying number of specified ratings with heuristic adjustment factors. For example, the right-hand side of Equation 2.29 uses an adjustment factor of $\sqrt{|Q_t(u)|}$ in the denominator. On the other hand, no such adjustment factor is used in the *SLIM* method. The adjustment issue is less pressing for the case of unary data sets, in which the presence of an item is usually the only information available. In such cases, replacing missing values with 0s is a common practice, and the bias of doing so is much lower than in the case where ratings indicate varying levels of likes or dislikes.

Therefore, the models share a number of conceptual similarities, although there are some differences at the detailed level.

2.7 Graph Models for Neighborhood-Based Methods

The sparsity of observed ratings causes a major problem in the computation of similarity in neighborhood-based methods. A number of graph models are used in order to define similarity in neighborhood-based methods, with the use of either structural transitivity or ranking techniques. Graphs are a powerful abstraction that enable many algorithmic tools from the network domain. The graphs provide a structural representation of the relationships among various users and/or items. The graphs can be constructed on the users, on the items, or on both. These different types of graphs result in a wide variety of algorithms, which use

[7] It is noteworthy that imposing an additional constraint, such as non-negativity, always reduces the quality of the optimal solution on the *observed* entries. On the other hand, imposing constraints increases the model bias and reduces model variance, which might reduce overfitting on the *unobserved* entries. In fact, when two closely related models have contradicting relative performances on the observed and unobserved entries, respectively, it is almost always a result of differential levels of overfitting in the two cases. You will learn more about the bias-variance trade-off in Chapter 6. In general, it is more reliable to predict item ratings with positive item-item relationships rather than negative relationships. The non-negativity constraint is based on this observation. The incorporation of model biases in the form of such natural constraints is particularly useful for smaller data sets.

either random-walk or shortest-path methods for recommendation. In the following, we will describe the algorithms used for performing recommendations with various types of graph representations of ratings matrices.

2.7.1 User-Item Graphs

It is possible to use structural measures on the *user-item graph*, rather than the Pearson correlation coefficient, for defining neighborhoods. Such an approach is more effective for sparse ratings matrices because one can use structural transitivity of edges for the recommendation process.

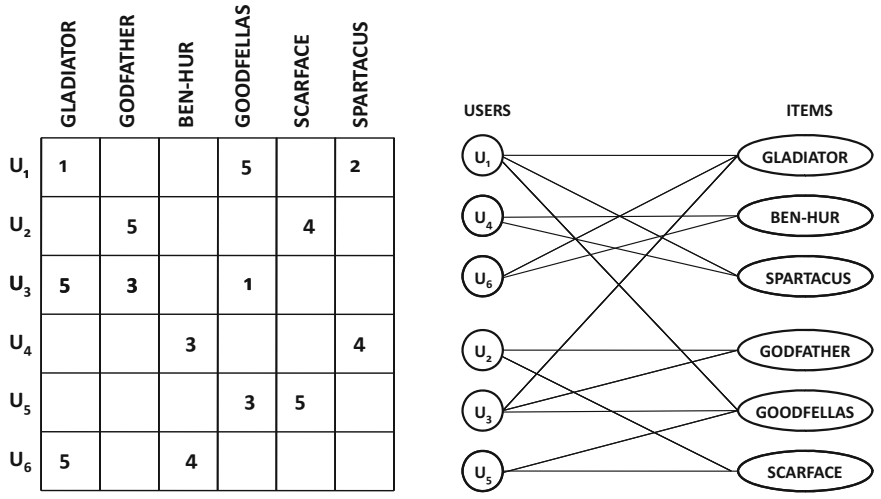
The user-item graph is defined as an undirected and bipartite graph $G = (N_u \cup N_i, A)$, where N_u is the set of nodes representing users, and N_i is the set of nodes representing items. All edges in the graph exist only between users and items. An undirected edge exists in A between a user i and an item j, if and only if user i has rated item j. Therefore, the number of edges is equal to the number of observed entries in the utility matrix. For example, the user-item graph for the ratings matrix of Figure 2.3(a) is illustrated in Figure 2.3(b). The main advantage of graph-based methods is that two users do not need to have rated many of the same items to be considered neighbors as long as many short paths exist between the two users. Therefore, this definition allows the construction of neighborhoods with the notion of *indirect* connectivity between nodes. Of course, if two users have rated many common items, then such a definition will also consider them close neighbors. Therefore, the graph-based approach provides a different way of defining neighborhoods, which can be useful in sparse settings.

The notion of indirect connectivity is achieved with the use of path- or walk-based definitions. Some common methods for achieving this goal include the use of random-walk measures or the *Katz measure*, which is discussed in section 2.7.1.2. Both these measures are closely related to the problem of *link prediction* in social network analysis (cf. section 10.4 of Chapter 10), and they demonstrate the fact that graphical models of recommender systems connect the link-prediction problem to the vanilla recommendation problem. In the following, we discuss different ways of defining neighborhoods on the graph representation.

2.7.1.1 Defining Neighborhoods with Random Walks

The neighborhood of a user is defined by the set of users that are encountered frequently in a random walk starting at that user. How can the expected frequency of such random walks be measured? The answer to this problem is closely related to the random-walk methods, which are used frequently in Web-ranking applications. One can use either the personalized *PageRank* or the *SimRank* method (cf. Chapter 10) to determine the k most similar users to a given user for user-based collaborative filtering. Similarly, one can use this method to determine the k most similar items to a given item by starting the random walk at a given *item*. This approach is useful for item-based collaborative filtering. The other steps of user-based collaborative filtering and item-based collaborative filtering remain the same.

Why is this approach more effective for sparse matrices? In the case of the Pearson's correlation coefficient, two users need to be connected *directly* to a set of common items for the neighborhood to be defined meaningfully. In sparse user-item graphs, such direct connectivity may not exist for many nodes. On the other hand, a random-walk method also considers *indirect* connectivity, because a walk from one node to another may use any number of steps. Therefore, as long as large portions of the user-item graphs are connected,



(a) Ratings matrix (b) User-item graphs of specified ratings

Figure 2.3: A ratings matrix and corresponding user-item graph

it is always possible to meaningfully define neighborhoods. Such user-item graphs can also be used to directly predict ratings with the use of a variety of models. Such related methods will be discussed in section 10.2.3.3 of Chapter 10.

2.7.1.2 Defining Neighborhoods with the Katz Measure

Rather than using a probabilistic measure, such as random walks, it is possible to use the weighted number of walks between a pair of nodes in order to determine the affinity between them. The weight of each walk is a discount factor in $(0, 1)$, which is typically a decreasing function of its length. The weighted number of walks between a pair of nodes is referred to as the *Katz* measure. The weighted number of walks between a pair of nodes is often used as a link-prediction measure. The intuition is that if two users belong to the same neighborhood based on walk-based connectivity, then there is a propensity for a link to be formed between them in the user-item graph. The specific level of propensity is measured with the number of (discounted) walks between them.

Definition 2.7.1 (Katz Measure) *Let $n_{ij}^{(t)}$ be the number of walks of length t between nodes i and j. Then, for a user-defined parameter $\beta < 1$, the Katz measure between nodes i and j is defined as follows:*

$$Katz(i, j) = \sum_{t=1}^{\infty} \beta^t \cdot n_{ij}^{(t)} \tag{2.34}$$

The value of β is a discount factor that de-emphasizes walks of longer lengths. For small enough values of β, the infinite summation of Equation 2.34 will converge.

Let K be the $m \times m$ matrix of Katz coefficients between pairs of users. If A is the symmetric adjacency matrix of an undirected network, then the pairwise Katz coefficient matrix K can be computed as follows:

$$K = \sum_{i=1}^{\infty} (\beta A)^i = (I - \beta A)^{-1} - I \tag{2.35}$$

The value of β should always be selected to be smaller than the inverse of the largest eigenvalue of A to ensure convergence of the infinite summation. The Katz measure is closely rated to diffusion kernels in graphs. In fact, several collaborative recommendation methods directly use diffusion kernels to make recommendations [205].

A weighted version of the measure can be computed by replacing A with the weight matrix of the graph. This can be useful in cases where one wishes to weight the edges in the user-item graph with the corresponding rating. The top-k nodes with the largest Katz measures to the target node are isolated as its neighborhood. Once the neighborhood has been determined, it is used to perform the prediction according to Equation 2.4. Many variations of this basic principle are used to make recommendations:

1. It is possible to use a threshold on the maximum path length in Equation 2.34. This is because longer path lengths generally become noisy for the prediction process. Nevertheless, because of the use of the discount factor β, the impact of long paths on the measure is generally limited.

2. In the aforementioned discussion, the Katz measure is used only to determine the neighborhoods of users. Therefore, the Katz measure is used to compute the affinity between pairs of *users*. After the neighborhood of a user has been determined, it is used to make predictions in the same way as any other neighborhood-based method.

   However, a different way of *directly* performing the prediction, without using neighborhood methods, would be to measure the affinity between users and *items*. The Katz measure can be used to compute these affinities. In such cases, the links are weighted with ratings, and the problem is reduced to that of predicting links between users and items. These methods will be discussed in more detail in section 10.4.6 of Chapter 10.

The bibliographic notes contain a number of references to various path-based methods.

2.7.2 User-User Graphs

In user-item graphs, the user-user connectivity is defined by an even number of hops in the user-item graph. Instead of constructing user-item graphs, one might instead directly create user-user graphs based on 2-hop connectivity between users. The advantage of user-user graphs over user-item graphs is that the edges of the graph are more informative in the former. This is because the 2-hop connectivity can directly take the number and similarity of common items between the two users into account, while creating the edges. These notions, referred to as *horting* and *predictability*, will be discussed slightly later. The algorithm uses the notion of horting to quantify the number of mutually specified ratings between two users (nodes), whereas it uses the notion of predictability to quantify the level of similarity among these common ratings.

The user-user graph is constructed as follows. Each node u corresponds to one of the m users in the $m \times n$ user-item matrix. Let I_u be the set of items for which ratings have been specified by user u, and let I_v be the set of items for which ratings have been specified by user v. Edges are defined in this graph with the notion of *horting*. Horting is an asymmetric relationship between users, which is defined on the basis of their having rated similar items.

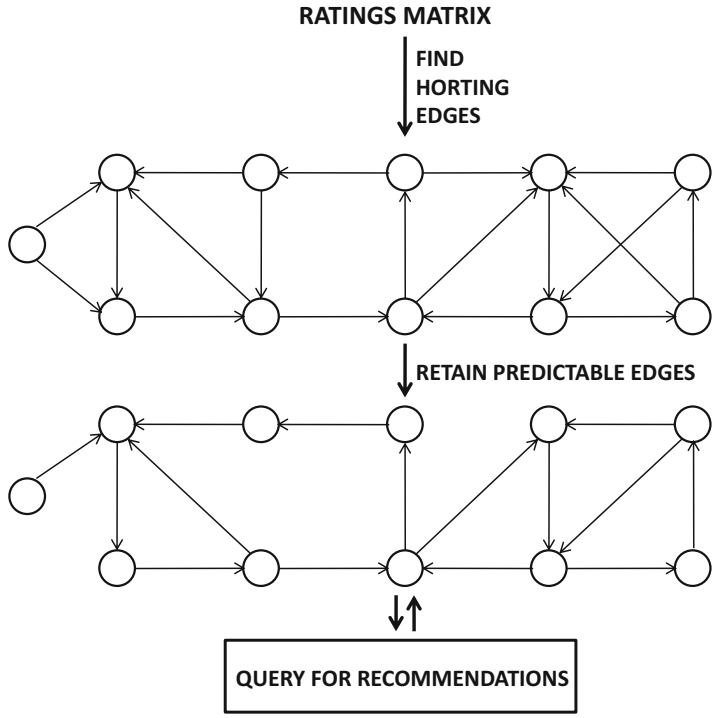


Figure 2.4: The user-user predictability approach

Definition 2.7.2 (Horting) *A user u is said to hort user v at level (F, G), if either of the following are true:*

$$|I_u \cap I_v| \geq F$$
$$|I_u \cap I_v|/|I_u| \geq G$$

Here, F and G are algorithm parameters. Note that it is sufficient for one of the two aforementioned conditions to hold for user u to hort user v. The notion of horting is used to further define predictability.

Definition 2.7.3 (Predictability) *The user v predicts user u, if u horts v and there exists a linear transformation function $f(\cdot)$ such that the following is true:*

$$\frac{\sum_{k \in I_u \cap I_v} |r_{uk} - f(r_{vk})|}{|I_u \cap I_v|} \leq U$$

Here, U is another algorithm parameter. It is noteworthy that the distance $\frac{\sum_{k \in I_u \cap I_v} |r_{uk} - f(r_{vk})|}{|I_u \cap I_v|}$ between the ratings of user u and the transformed ratings of user v is a variant of the Manhattan distance on their common specified ratings. The main difference from the Manhattan distance is that the distance is normalized by the number of mutually specified ratings between the two users. This distance is also referred to as the *Manhattan segmental distance*.

The directions of horting and predictability are opposite one another. In other words, for user v to predict user u, u must hort v. A directed graph G is defined, in which an edge exists from u to v, if v predicts u. This graph is referred to as the *user-user predictability graph*. Each edge in this graph corresponds to a linear transformation, as discussed in Definition 2.7.3. The linear transformation defines a prediction, where the rating at the head of the edge can be used to predict the rating at the tail of the edge. Furthermore, it is assumed that one can apply these linear transformations in a transitive way over a directed path in order to predict the rating of the source of the path from the rating at the destination of the path.

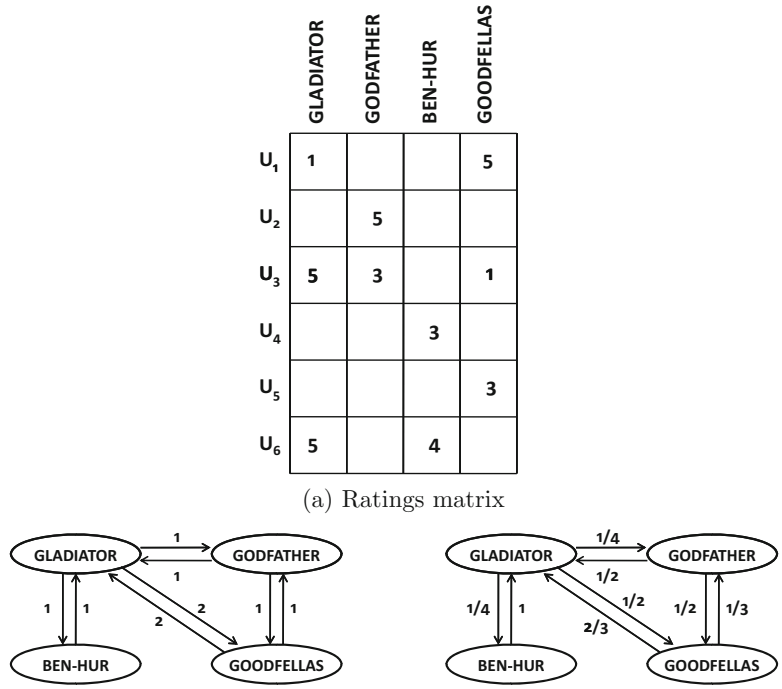
Then, the rating of a target user u for an item k is computed by determining all the directed shortest paths from user u to all other users who have rated item k. Consider a directed path of length r from user u to a user v who has rated item k. Let $f_1 \ldots f_r$ represent the sequence of linear transformations along the directed path starting from node u to this user v. Then, the rating prediction $\hat{r}_{uk}^{(v)}$ of the rating of target user u for item k (based only on user v) is given by applying the composition of the r linear mappings along this path from user u to v, to the rating r_{vk} of user v on item k:

$$\hat{r}_{uk}^{(v)} = (f_1 \circ f_2 \ldots \circ f_r)(r_{vk}) \tag{2.36}$$

The rating prediction $\hat{r}_{uk}^{(v)}$ contains the superscript v because it is based only on the rating of user v. Therefore, the final rating prediction $\hat{r}_{uk}$ is computing by averaging the value of $\hat{r}_{uk}^{(v)}$ over all users v that have rated item k, within a threshold distance D of the target user u.

Given a target user (node) u, one only needs to determine directed *paths* from this user to other users, who have rated the item at hand. The shortest path can be determined with the use of a breadth-first algorithm, which is quite efficient. Another important detail is that a threshold is imposed on the maximum path length that is usable for prediction. If no user, who has rated item k is found within a threshold length D of the target node u, then the algorithm terminates with failure. In other words, the rating of the target user u for item k simply cannot be determined robustly with the available ratings matrix. It is important to impose such thresholds to improve efficiency and also because the linear transformation along very long path lengths might lead to increasing distortion in the rating prediction. The overall approach is illustrated in Figure 2.4. Note that a directed edge exists from u to v in the horting graph if u horts v. On the other hand, an edge exists in the predictability graph if u horts v *and* v predicts u. Therefore, the predictability graph is obtained from the horting graph by dropping a few edges. This graph is set up in an offline phase and it is repeatedly queried for recommendations. In addition, a number of index data structures are set up from the ratings matrix during the offline setup phase. These data structures are used along with the predictability graph in order to resolve the queries efficiently. More details on the horting approach may be found in [33].

This approach can work for very sparse matrices because it uses transitivity to predict ratings. An important challenge in neighborhood methods is the lack of coverage of rating prediction. For example, if none of John's immediate neighbors have rated *Terminator*, it is impossible to provide a rating prediction for John. However, structural transitivity allows us to check whether the *indirect* neighbors of John have rated *Terminator*. Therefore, the main advantage of this approach is that it has better coverage compared to competing methods.



(a) Ratings matrix

(b) Unnormalized correlation graph

(c) Normalized correlation graph

Figure 2.5: A ratings matrix and its correlation graphs

2.7.3 Item-Item Graphs

It is also possible to leverage item-item graphs to perform the recommendations. Such a graph is also referred to as the *correlation graph* [232]. In this case, a weighted and directed network $G = (N, A)$ is constructed, in which each node in N corresponds to an item, and each edge in A corresponds to a relationship between items. The weight w_{ij} is associated with each edge (i, j). If items i and j have both been rated by at least one common user, then both the directed edges (i, j) and (j, i) exist in the network. Otherwise, no edges exist between nodes i and j. The directed network is, however, asymmetric because the weight of edge (i, j) is not necessarily the same as that of edge (j, i). Let U_i be the set of users that have specified ratings for item i and U_j be the set of users that have specified ratings for item j. Then, the weight of the edge (i, j) is computed using the following simple algorithm.

First, we initialize the weight w_{ij} of each edge (i, j) to $|U_i \cap U_j|$. At this point, the edge weights are symmetric because $w_{ij} = w_{ji}$. Then, the weights of the edges are normalized, so that the sum of the weights of the outgoing edges of a node is equal to 1. This normalization is achieved by dividing w_{ij} with the sum of the outgoing weights from node i. The normalization step results in asymmetric weights, because each of the weights w_{ij} and w_{ji} are divided by different quantities. This results in a graph in which the weights on edges correspond to *random-walk probabilities*. An example of the correlation graph for a ratings matrix is illustrated in Figure 2.5. It is clear that the weights on the *normalized* correlation graph are not symmetric because of the scaling of the weights to transition probabilities. Furthermore, it is noteworthy that the rating values are not used in the construction of the correlation graph. Only the *number* of observed ratings in common between two items

is used. This is sometimes not desirable. It is, of course, possible to define the correlation graph in other ways, such as with the use of the cosine function between the *rating* vectors of the two items.

As discussed in Chapter 10, random-walk methods can be used to determine the neighborhood of a given item. The resulting neighborhood can be used for item-based collaborative filtering methods. Furthermore, personalized *PageRank* methods can be used to directly determine the ratings on the item-item graph. This method is referred to as *ItemRank*, and it is discussed in section 10.2.3.3 of Chapter 10.

2.8 Summary

Because collaborative filtering can be viewed as a generalization of classification and regression problems, the methodologies for the latter classes of problems can also be applied to the former. Neighborhood-based methods derive their inspiration from nearest neighbor classification and regression methods. In user-based methods, the first step is to determine the neighborhood of the target user. In order to compute the neighborhood, a variety of similarity functions, such as the Pearson correlation coefficient or the cosine, are used. The neighborhood is used in order to extrapolate the unknown ratings of a record. In item-based methods, the most similar items are computed with respect to a target item. Then, the user's own ratings on these similar items are used in order to make a rating prediction. Item-based methods are likely to have more relevant recommendations, but they are less likely to yield diverse recommendations. In order to speed up neighborhood-based methods, clustering is often used.

Neighborhood-based methods can be viewed as linear models, in which the weights are chosen in a heuristic way with the use of similarity values. One can also learn these weights with the use of linear regression models. Such methods have the advantage that they can be combined with other optimization models, such as matrix factorization, for better prediction. Such methods are discussed in the next chapter.

Neighborhood-based methods face numerous challenges because of data sparsity. Users often specify only a small number of ratings. As a result, a pair of users may often have specified only a small number of ratings. Such situations can be addressed effectively with the use of both dimensionality reduction and graph-based models. While dimensionality reduction methods are often used as standalone methods for collaborative filtering, they can also be combined with neighborhood-based methods to improve the effectiveness and efficiency of collaborative filtering. Various types of graphs can be extracted from rating patterns, such as user-item graphs, user-user graphs, or item-item graphs. Typically, random-walk or shortest-path methods are used in these cases.

2.9 Bibliographic Notes

Neighborhood-based methods were among the earliest techniques used in the field of recommender systems. The earliest user-based collaborative filtering models were studied in [33, 98, 501, 540]. A comprehensive survey of neighborhood-based recommender systems may be found in [183]. Sparsity is a major problem in such systems, and various graph-based systems have been designed to alleviate the problem of sparsity [33, 204, 647]. Methods that are specifically designed for the long tail in recommender algorithms are discussed in [173, 463, 648].

User-based methods utilize the ratings of *similar* users on the *same* item in order to make predictions. While such methods were initially quite popular, they are not easily scalable and sometimes inaccurate. Subsequently, item-based methods [181, 360, 524] were proposed, which compute predicted ratings as a function of the ratings of the *same* user on *similar* items. Item-based methods provide more accurate but less diverse recommendations.

The notion of mean-centering for improving recommendation algorithms was proposed in [98, 501]. A comparison of the use of the Z-score with mean-centering is studied in [245, 258], and these two studies provide somewhat conflicting results. A number of methods which do not use the absolute ratings, but instead focus on ordering the ratings in terms of preference weights, are discussed in [163, 281, 282]. The significance-weighting methods of de-emphasizing the neighbors who have too few common ratings with a given neighbor are discussed in [71, 245, 247, 380]. Many different variants of the similarity function are used for computing the neighbor. Two such examples are the *mean-squared distance* [540] and the *Spearman rank correlation* [299]. The specific advantage of these distance measures is not quite clear because conflicting results have been presented in the literature [247, 258]. Nevertheless, the consensus seems to be that the Pearson rank correlation provides the most accurate results [247]. Techniques for adjusting for the impact of very popular items are discussed in [98, 280]. The use of exponentiated amplification for prediction in neighborhood-based methods is discussed in [98]. A discussion of the use of voting techniques in nearest neighbor methods may be found in [183]. Voting methods can be viewed as a direct generalization of the nearest neighbor classifier, as opposed to a generalization of nearest neighbor regression modeling.

Methods for item-based collaborative filtering were proposed in [181, 524, 526]. A detailed study of different variations of item-based collaborative filtering algorithms is provided in [526], along with a comparison with respect to user-based methods. The item-based method in [360] is notable because it describes one of Amazon.com's collaborative filtering methods. The user-based and item-based collaborative filtering methods have also been unified with the notion of similarity fusion [622]. A more generic unification framework may be found in [613]. Clustering methods are used frequently to improve the efficiency of neighborhood-based collaborative filtering. A number of clustering methods are described in [146, 167, 528, 643, 644, 647]. The extension of neighborhood methods to very large-scale data sets has been studied in [51].

Dimensionality reduction techniques have a rich history of being used in missing-value estimation [24, 472] and recommender systems [71, 72, 228, 252, 309, 313, 500, 517, 525]. In fact, most of these techniques directly use such latent models to predict the ratings without relying on neighborhood models. However, some of these dimensionality reduction techniques [71, 72, 309, 525] are specifically designed to improve the effectiveness and efficiency of neighborhood-based techniques. A key contribution of [72] is to provide an insight about the relationship between neighborhood methods and regression-based methods. This relationship is important because it shows how one can formulate neighborhood-based methods as model-based methods with a crisp optimization formulation. Note that many other model-based methods, such as latent factor models, can also be expressed as optimization formulations. This observation paves the way for combining neighborhood methods with latent factor models in a unified framework [309] because one can now combine the two objective functions. Other regression-based models for recommender systems, such as *slope-one predictors* and ordinary least-squares methods, are proposed in [342, 620]. Methods for learning pairwise preferences over itemsets are discussed in [469]. Item-item regression models have also been studied in the context of *Sparse Linear Models (SLIM)* [455], where an elastic-net regularizer is used on the linear model without restricting the coefficients to the

neighborhood of the item. Higher-order sparse learning methods, which model the effects of using combinations of items, are discussed in [159]. Efficient methods for training linear models and tuning regularization parameters are discussed in [347]. Constrained linear regression methods are discussed in [430].

A general examination of linear classifiers, such as least-squares regression and support vector machines, is provided in [669]. However, the approach is designed for implicit feedback data sets in which only positive preferences are specified. It was observed that collaborative filtering, in such cases, is similar to text categorization. However, because of the noise in the data and the imbalanced nature of the class distribution, a direct use of SVM methods is sometimes not effective. Changes to the loss function are suggested in [669] in order to provide more accurate results.

Many graph-based methods have been proposed for improving collaborative filtering algorithms. Most of these methods are based on either user-item graphs, but a few are also based on user-user graphs. An important observation from the perspective of graph-based methods is that they show an interesting relationship between the problems of ranking, recommendation, and link-prediction. The use of random walks for determining the neighborhood in recommendation systems is discussed in [204, 647]. A method, which uses the *number* of discounted paths between a pair of nodes in a user-item graph for recommendations, was proposed in [262]. This approach is equivalent to using the Katz measure between user-user pairs in order to determine whether they reside in each other's neighborhoods. This approach is related to link-prediction [354], because the Katz measure is often used to determine the linkage affinity between a pair of nodes. A survey on link prediction methods may be found in [17]. Some graph-based methods do not directly use neighborhoods. For example, the *ItemRank* method proposed in [232] shows how to use ranking directly to make predictions, and the method in [261] shows how to use link-prediction methods directly for collaborative filtering. These methods are also discussed in Chapter 10 of this book. Techniques for leveraging user-user graphs are discussed in [33]. These methods have the advantage that they directly encode the user-user similarity relationships in the edges of the graph. As a result, the approach provides better coverage than competing methods.

2.10 Exercises

1. Consider the ratings matrix of Table 2.1. Predict the absolute rating of item 3 for user 2 using:

(a) User-based collaborative filtering with Pearson correlation and mean-centering

(b) Item-based collaborative filtering with adjusted cosine similarity

Use a neighborhood of size 2 in each case.

2. Consider the following ratings table between five users and six items:

Item-Id ⇒	1	2	3	4	5	6
1	5	6	7	4	3	?
2	4	?	3	?	5	4
3	?	3	4	1	1	?
4	7	4	3	6	?	4
5	1	?	3	2	2	5

(a) Predict the values of unspecified ratings of user 2 using user-based collaborative filtering algorithms. Use the Pearson correlation with mean-centering.

(b) Predict the values of unspecified ratings of user 2 using item-based collaborative filtering algorithms. Use the adjusted cosine similarity.

Assume that a peer group of size at most 2 is used in each case, and negative correlations are filtered out.

3. Discuss the similarity between a k-nearest neighbor classifier in traditional machine learning and the user-based collaborative filtering algorithm. Describe an analogous classifier to item-based collaborative filtering.

4. Consider an algorithm that performs clustering of users based on their ratings matrix and reports the average ratings within a cluster as the predicted items ratings for every user within a cluster. Discuss the effectiveness and efficiency trade-offs of such an approach compared to a neighborhood model.

5. Propose an algorithm that uses random walks on a user-user graph to perform neighborhood-based collaborative filtering. [This question requires a background in ranking methods.]

6. Discuss various ways in which graph clustering algorithms can be used to perform neighborhood-based collaborative filtering.

7. Implement the user-based and item-based collaborative filtering algorithms.

8. Suppose you had content-based profiles associated with users indicating their interests and profiles associated with items corresponding to their descriptions. At the same time, you had a ratings matrix between users and items. Discuss how you can incorporate the content-based information within the framework of graph-based algorithms.

9. Suppose that you had a unary ratings matrix. Show how collaborative filtering algorithms can be solved using content-based methods by treating the ratings of an item as its features. Refer to Chapter 1 for a description of content-based methods. What type of a content-based classifier does an item-based collaborative filtering algorithm correspond to?

http://www.springer.com/978-3-319-29657-9

Recommender Systems
The Textbook
Aggarwal, C.C.
2016, XXI, 498 p. 79 illus., 18 illus. in color., Hardcover
ISBN: 978-3-319-29657-9